

# Tasks

This section of the Kubernetes documentation contains pages that show how to do individual tasks. A task page shows how to do a single thing, typically by giving a short sequence of steps.

If you would like to write a task page, see [Creating a Documentation Pull Request](#).

---

## [Install Tools](#)

Set up Kubernetes tools on your computer.

## [Administer a Cluster](#)

Learn common tasks for administering a cluster.

## [Configure Pods and Containers](#)

Perform common configuration tasks for Pods and containers.

## [Manage Kubernetes Objects](#)

Declarative and imperative paradigms for interacting with the Kubernetes API.

## [Managing Secrets](#)

Managing confidential settings data using Secrets.

## [Inject Data Into Applications](#)

Specify configuration and other data for the Pods that run your workload.

## [Run Applications](#)

Run and manage both stateless and stateful applications.

## [Run Jobs](#)

Run Jobs using parallel processing.

## [Access Applications in a Cluster](#)

Configure load balancing, port forwarding, or setup firewall or DNS configurations to access applications in a cluster.

## [Monitoring, Logging, and Debugging](#)

Set up monitoring and logging to troubleshoot a cluster, or debug a containerized application.

## [Extend Kubernetes](#)

Understand advanced ways to adapt your Kubernetes cluster to the needs of your work environment.

### [TLS](#)

Understand how to protect traffic within your cluster using Transport Layer Security (TLS).

## [Manage Cluster Daemons](#)

Perform common tasks for managing a DaemonSet, such as performing a rolling update.

## [Service Catalog](#)

Install the Service Catalog extension API.

## [Networking](#)

Learn how to configure networking for your cluster.

## [Configure a kubelet image credential provider](#)

Configure the kubelet's image credential provider plugin

## [Extend kubectl with plugins](#)

Extend kubectl by creating and installing kubectl plugins.

## [Manage HugePages](#)

Configure and manage huge pages as a schedulable resource in a cluster.

## [Schedule GPUs](#)

Configure and schedule GPUs for use as a resource by nodes in a cluster.

# **Install Tools**

Set up Kubernetes tools on your computer.

# kubectl

The Kubernetes command-line tool, `kubectl`, allows you to run commands against Kubernetes clusters. You can use `kubectl` to deploy applications, inspect and manage cluster resources, and view logs.

See [Install and Set Up kubectl](#) for information about how to download and install `kubectl` and set it up for accessing your cluster.

[View kubectl Install and Set Up Guide](#)

You can also read the [kubectl reference documentation](#).

# kind

`kind` lets you run Kubernetes on your local computer. This tool requires that you have [Docker](#) installed and configured.

The kind [Quick Start](#) page shows you what you need to do to get up and running with kind.

[View kind Quick Start Guide](#)

# minikube

Like kind, `minikube` is a tool that lets you run Kubernetes locally. `minikube` runs a single-node Kubernetes cluster on your personal computer (including Windows, macOS and Linux PCs) so that you can try out Kubernetes, or for daily development work.

You can follow the official [Get Started!](#) guide if your focus is on getting the tool installed.

[View minikube Get Started! Guide](#)

Once you have `minikube` working, you can use it to [run a sample application](#).

# kubeadm

You can use the `kubeadm` tool to create and manage Kubernetes clusters. It performs the actions necessary to get a minimum viable, secure cluster up and running in a user friendly way.

[Installing kubeadm](#) shows you how to install `kubeadm`. Once installed, you can use it to [create a cluster](#).

[View kubeadm Install Guide](#)

# Install and Set Up kubectl

The Kubernetes command-line tool, [kubectl](#), allows you to run commands against Kubernetes clusters. You can use kubectl to deploy applications, inspect and manage cluster resources, and view logs. For a complete list of kubectl operations, see [Overview of kubectl](#).

## Before you begin

You must use a kubectl version that is within one minor version difference of your cluster. For example, a v1.2 client should work with v1.1, v1.2, and v1.3 master. Using the latest version of kubectl helps avoid unforeseen issues.

## Install kubectl on Linux

### Install kubectl binary with curl on Linux

1. Download the latest release with the command:

```
curl -LO "https://storage.googleapis.com/kubernetes-release/release/$(curl -s https://storage.googleapis.com/kubernetes-release/release/stable.txt)/bin/linux/amd64/kubectl"
```

To download a specific version, replace the `$(curl -s https://storage.googleapis.com/kubernetes-release/release/stable.txt)` portion of the command with the specific version.

For example, to download version v1.20.0 on Linux, type:

```
curl -LO https://storage.googleapis.com/kubernetes-release/release/v1.20.0/bin/linux/amd64/kubectl
```

2. Make the kubectl binary executable.

```
chmod +x ./kubectl
```

3. Move the binary in to your PATH.

```
sudo mv ./kubectl /usr/local/bin/kubectl
```

4. Test to ensure the version you installed is up-to-date:

```
kubectl version --client
```

### Install using native package management

- [Ubuntu, Debian or HypriotOS](#)
- [CentOS, RHEL or Fedora](#)

```
sudo apt-get update && sudo apt-get install -y apt-transport-  
https gnupg2 curl  
curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg |  
sudo apt-key add -  
echo "deb https://apt.kubernetes.io/ kubernetes-xenial main" |  
sudo tee -a /etc/apt/sources.list.d/kubernetes.list  
sudo apt-get update  
sudo apt-get install -y kubectl
```

```
cat <<EOF > /etc/yum.repos.d/kubernetes.repo  
[kubernetes]  
name=Kubernetes  
baseurl=https://packages.cloud.google.com/yum/repos/kubernetes-  
el7-x86_64  
enabled=1  
gpgcheck=1  
repo_gpgcheck=1  
gpgkey=https://packages.cloud.google.com/yum/doc/yum-key.gpg  
https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg  
EOF  
yum install -y kubectl
```

## Install using other package management

- [Snap](#)
- [Homebrew](#)

If you are on Ubuntu or another Linux distribution that support [snap](#) package manager, kubectl is available as a [snap](#) application.

```
snap install kubectl --classic  
kubectl version --client
```

If you are on Linux and using [Homebrew](#) package manager, kubectl is available for [installation](#).

```
brew install kubectl  
kubectl version --client
```

## Install kubectl on macOS

### Install kubectl binary with curl on macOS

1. Download the latest release:

```
curl -LO "https://storage.googleapis.com/kubernetes-release/  
release/$(curl -s https://storage.googleapis.com/kubernetes-  
release/release/stable.txt)/bin/darwin/amd64/kubectl"
```

To download a specific version, replace the `$(curl -s https://storage.googleapis.com/kubernetes-release/release/stable.txt)` portion of the command with the specific version.

For example, to download version v1.20.0 on macOS, type:

```
curl -LO https://storage.googleapis.com/kubernetes-release/release/v1.20.0/bin/darwin/amd64/kubectl
```

Make the kubectl binary executable.

```
chmod +x ./kubectl
```

2. Move the binary in to your PATH.

```
sudo mv ./kubectl /usr/local/bin/kubectl
```

3. Test to ensure the version you installed is up-to-date:

```
kubectl version --client
```

## Install with Homebrew on macOS

If you are on macOS and using [Homebrew](#) package manager, you can install kubectl with Homebrew.

1. Run the installation command:

```
brew install kubectl
```

or

```
brew install kubernetes-cli
```

2. Test to ensure the version you installed is up-to-date:

```
kubectl version --client
```

## Install with Macports on macOS

If you are on macOS and using [Macports](#) package manager, you can install kubectl with Macports.

1. Run the installation command:

```
sudo port selfupdate  
sudo port install kubectl
```

2. Test to ensure the version you installed is up-to-date:

```
kubectl version --client
```

# Install kubectl on Windows

## Install kubectl binary with curl on Windows

1. Download the latest release v1.20.0 from [this link](#).

Or if you have curl installed, use this command:

```
curl -LO https://storage.googleapis.com/kubernetes-release/release/v1.20.0/bin/windows/amd64/kubectl.exe
```

To find out the latest stable version (for example, for scripting), take a look at <https://storage.googleapis.com/kubernetes-release/release/stable.txt>.

2. Add the binary in to your PATH.
3. Test to ensure the version of kubectl is the same as downloaded:

```
kubectl version --client
```

**Note:** [Docker Desktop for Windows](#) adds its own version of kubectl to PATH. If you have installed Docker Desktop before, you may need to place your PATH entry before the one added by the Docker Desktop installer or remove the Docker Desktop's kubectl.

## Install with Powershell from PSGallery

If you are on Windows and using [Powershell Gallery](#) package manager, you can install and update kubectl with Powershell.

1. Run the installation commands (making sure to specify a DownloadLocation):

```
Install-Script -Name 'install-kubectl' -Scope CurrentUser -Force  
install-kubectl.ps1 [-DownloadLocation <path>]
```

**Note:** If you do not specify a DownloadLocation, kubectl will be installed in the user's temp Directory.

The installer creates \$HOME/.kube and instructs it to create a config file.

2. Test to ensure the version you installed is up-to-date:

```
kubectl version --client
```

**Note:** Updating the installation is performed by rerunning the two commands listed in step 1.

## Install on Windows using Chocolatey or Scoop

1. To install kubectl on Windows you can use either [Chocolatey](#) package manager or [Scoop](#) command-line installer.

- [choco](#)
- [scoop](#)

```
choco install kubernetes-cli
```

```
scoop install kubectl
```

2. Test to ensure the version you installed is up-to-date:

```
kubectl version --client
```

3. Navigate to your home directory:

```
# If you're using cmd.exe, run: cd %USERPROFILE%
cd ~
```

4. Create the .kube directory:

```
mkdir .kube
```

5. Change to the .kube directory you just created:

```
cd .kube
```

6. Configure kubectl to use a remote Kubernetes cluster:

```
New-Item config -type file
```

**Note:** Edit the config file with a text editor of your choice, such as Notepad.

## Download as part of the Google Cloud SDK

You can install kubectl as part of the Google Cloud SDK.

1. Install the [Google Cloud SDK](#).
2. Run the kubectl installation command:

```
gcloud components install kubectl
```

3. Test to ensure the version you installed is up-to-date:

```
kubectl version --client
```

# Verifying kubectl configuration

In order for kubectl to find and access a Kubernetes cluster, it needs a [kubeconfig file](#), which is created automatically when you create a cluster using [kube-up.sh](#) or successfully deploy a Minikube cluster. By default, kubectl configuration is located at `~/.kube/config`.

Check that kubectl is properly configured by getting the cluster state:

```
kubectl cluster-info
```

If you see a URL response, kubectl is correctly configured to access your cluster.

If you see a message similar to the following, kubectl is not configured correctly or is not able to connect to a Kubernetes cluster.

```
The connection to the server <server-name:port> was refused -  
did you specify the right host or port?
```

For example, if you are intending to run a Kubernetes cluster on your laptop (locally), you will need a tool like Minikube to be installed first and then re-run the commands stated above.

If kubectl cluster-info returns the url response but you can't access your cluster, to check whether it is configured properly, use:

```
kubectl cluster-info dump
```

## Optional kubectl configurations

### Enabling shell completion

kubectl provides autocomplete support for Bash and Zsh, which can save you a lot of typing.

Below are the procedures to set up autocomplete for Bash (including the difference between Linux and macOS) and Zsh.

- [Bash on Linux](#)
- [Bash on macOS](#)
- [Zsh](#)

### Introduction

The kubectl completion script for Bash can be generated with the command `kubectl completion bash`. Sourcing the completion script in your shell enables kubectl autocomplete.

However, the completion script depends on [bash-completion](#), which means that you have to install this software first (you can test if you have bash-completion already installed by running `type _init_completion`).

## Install bash-completion

bash-completion is provided by many package managers (see [here](#)). You can install it with `apt-get install bash-completion` or `yum install bash-completion`, etc.

The above commands create `/usr/share/bash-completion/bash_completion`, which is the main script of bash-completion. Depending on your package manager, you have to manually source this file in your `~/.bashrc` file.

To find out, reload your shell and run `type _init_completion`. If the command succeeds, you're already set, otherwise add the following to your `~/.bashrc` file:

```
source /usr/share/bash-completion/bash_completion
```

Reload your shell and verify that bash-completion is correctly installed by typing `type _init_completion`.

## Enable kubectl autocompletion

You now need to ensure that the `kubectl` completion script gets sourced in all your shell sessions. There are two ways in which you can do this:

- Source the completion script in your `~/.bashrc` file:

```
echo 'source <(kubectl completion bash)' >>~/.bashrc
```

- Add the completion script to the `/etc/bash_completion.d` directory:

```
kubectl completion bash >/etc/bash_completion.d/kubectl
```

If you have an alias for `kubectl`, you can extend shell completion to work with that alias:

```
echo 'alias k=kubectl' >>~/.bashrc
echo 'complete -F __start_kubectl k' >>~/.bashrc
```

**Note:** bash-completion sources all completion scripts in `/etc/bash_completion.d`.

Both approaches are equivalent. After reloading your shell, `kubectl` autocompletion should be working.

## Introduction

The `kubectl` completion script for Bash can be generated with `kubectl completion bash`. Sourcing this script in your shell enables `kubectl` completion.

However, the `kubectl` completion script depends on [\*\*bash-completion\*\*](#) which you thus have to previously install.

**Warning:** There are two versions of bash-completion, v1 and v2. V1 is for Bash 3.2 (which is the default on macOS), and v2 is for Bash 4.1+. The kubectl completion script **doesn't work** correctly with bash-completion v1 and Bash 3.2. It requires **bash-completion v2** and **Bash 4.1+**. Thus, to be able to correctly use kubectl completion on macOS, you have to install and use Bash 4.1+ ([instructions](#)). The following instructions assume that you use Bash 4.1+ (that is, any Bash version of 4.1 or newer).

## Upgrade Bash

The instructions here assume you use Bash 4.1+. You can check your Bash's version by running:

```
echo $BASH_VERSION
```

If it is too old, you can install/upgrade it using Homebrew:

```
brew install bash
```

Reload your shell and verify that the desired version is being used:

```
echo $BASH_VERSION $SHELL
```

Homebrew usually installs it at `/usr/local/bin/bash`.

## Install bash-completion

**Note:** As mentioned, these instructions assume you use Bash 4.1+, which means you will install bash-completion v2 (in contrast to Bash 3.2 and bash-completion v1, in which case kubectl completion won't work).

You can test if you have bash-completion v2 already installed with `type _init_completion`. If not, you can install it with Homebrew:

```
brew install bash-completion@2
```

As stated in the output of this command, add the following to your `~/.bash_profile` file:

```
export BASH_COMPLETION_COMPAT_DIR="/usr/local/etc/
bash_completion.d"
[[ -r "/usr/local/etc/profile.d/bash_completion.sh" ]] && . "/
/usr/local/etc/profile.d/bash_completion.sh"
```

Reload your shell and verify that bash-completion v2 is correctly installed with `type _init_completion`.

## Enable kubectl autocompletion

You now have to ensure that the kubectl completion script gets sourced in all your shell sessions. There are multiple ways to achieve this:

- Source the completion script in your `~/.bash_profile` file:

```
echo 'source <(kubectl completion bash)' >>~/.bash_profile
```

- Add the completion script to the `/usr/local/etc/bash_completion.d` directory:

```
kubectl completion bash >/usr/local/etc/bash_completion.d/  
kubectl
```

- If you have an alias for kubectl, you can extend shell completion to work with that alias:

```
echo 'alias k=kubectl' >>~/.bash_profile  
echo 'complete -F __start_kubectl k' >>~/.bash_profile
```

- If you installed kubectl with Homebrew (as explained [above](#)), then the kubectl completion script should already be in `/usr/local/etc/bash_completion.d/kubectl`. In that case, you don't need to do anything.

**Note:** The Homebrew installation of bash-completion v2 sources all the files in the `BASH_COMPLETION_COMPAT_DIR` directory, that's why the latter two methods work.

In any case, after reloading your shell, kubectl completion should be working.

The kubectl completion script for Zsh can be generated with the command `kubectl completion zsh`. Sourcing the completion script in your shell enables kubectl autocompletion.

To do so in all your shell sessions, add the following to your `~/.zshrc` file:

```
source <(kubectl completion zsh)
```

If you have an alias for kubectl, you can extend shell completion to work with that alias:

```
echo 'alias k=kubectl' >>~/.zshrc  
echo 'complete -F __start_kubectl k' >>~/.zshrc
```

After reloading your shell, kubectl autocompletion should be working.

If you get an error like `complete:13: command not found: compdef`, then add the following to the beginning of your `~/.zshrc` file:

```
autoload -Uz compinit  
compinit
```

## What's next

- [Install Minikube](#)
- See the [getting started guides](#) for more about creating clusters.
- [Learn how to launch and expose your application](#).
- If you need access to a cluster you didn't create, see the [Sharing Cluster Access document](#).
- Read the [kubectl reference docs](#)

## 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](#).

Last modified October 22, 2020 at 3:01 PM PST: [Fix links in the tasks section \(740eb340d\)](#)

[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Install kubectl on Linux](#)
  - [Install kubectl binary with curl on Linux](#)
  - [Install using native package management](#)
  - [Install using other package management](#)
- [Install kubectl on macOS](#)
  - [Install kubectl binary with curl on macOS](#)
  - [Install with Homebrew on macOS](#)
  - [Install with Macports on macOS](#)
- [Install kubectl on Windows](#)
  - [Install kubectl binary with curl on Windows](#)
  - [Install with Powershell from PSGallery](#)
  - [Install on Windows using Chocolatey or Scoop](#)
- [Download as part of the Google Cloud SDK](#)
- [Verifying kubectl configuration](#)
- [Optional kubectl configurations](#)
  - [Enabling shell autocompletion](#)
- [What's next](#)

## Administer a Cluster

Learn common tasks for administering a cluster.

[\*\*Administration with kubeadm\*\*](#)

[\*\*Manage Memory, CPU, and API Resources\*\*](#)

[\*\*Install a Network Policy Provider\*\*](#)

[\*\*Access Clusters Using the Kubernetes API\*\*](#)

[\*\*Access Services Running on Clusters\*\*](#)

[\*\*Advertise Extended Resources for a Node\*\*](#)

[\*\*Autoscale the DNS Service in a Cluster\*\*](#)

[\*\*Change the default StorageClass\*\*](#)

[\*\*Change the Reclaim Policy of a PersistentVolume\*\*](#)

[\*\*Cloud Controller Manager Administration\*\*](#)

[\*\*Configure Out of Resource Handling\*\*](#)

[\*\*Configure Quotas for API Objects\*\*](#)

[\*\*Control CPU Management Policies on the Node\*\*](#)

[\*\*Control Topology Management Policies on a node\*\*](#)

[\*\*Customizing DNS Service\*\*](#)

[\*\*Debugging DNS Resolution\*\*](#)

[\*\*Declare Network Policy\*\*](#)

[\*\*Developing Cloud Controller Manager\*\*](#)

[\*\*Enable Or Disable A Kubernetes API\*\*](#)

[\*\*Enabling EndpointSlices\*\*](#)

[\*\*Enabling Service Topology\*\*](#)

[\*\*Encrypting Secret Data at Rest\*\*](#)

[\*\*Guaranteed Scheduling For Critical Add-On Pods\*\*](#)

[\*\*IP Masquerade Agent User Guide\*\*](#)

[\*\*Limit Storage Consumption\*\*](#)

[\*\*Namespaces Walkthrough\*\*](#)

[Operating etcd clusters for Kubernetes](#)

[Reconfigure a Node's Kubelet in a Live Cluster](#)

[Reserve Compute Resources for System Daemons](#)

[Safely Drain a Node](#)

[Securing a Cluster](#)

[Set Kubelet parameters via a config file](#)

[Set up High-Availability Kubernetes Masters](#)

[Share a Cluster with Namespaces](#)

[Upgrade A Cluster](#)

[Using a KMS provider for data encryption](#)

[Using CoreDNS for Service Discovery](#)

[Using NodeLocal DNSCache in Kubernetes clusters](#)

[Using sysctls in a Kubernetes Cluster](#)

## **Administration with kubeadm**

---

[Certificate Management with kubeadm](#)

[Upgrading kubeadm clusters](#)

[Adding Windows nodes](#)

[Upgrading Windows nodes](#)

## **Certificate Management with kubeadm**

**FEATURE STATE:** Kubernetes v1.15 [stable]

Client certificates generated by [kubeadm](#) expire after 1 year. This page explains how to manage certificate renewals with kubeadm.

# **Before you begin**

You should be familiar with [PKI certificates and requirements in Kubernetes](#).

## **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 the `certificatesDir` field of `kubeadm`'s `ClusterConfiguration`. By default this is `/etc/kubernetes/pki`.

If a given certificate and private key pair exists before running `kubeadm init`, `kubeadm` does not overwrite them. 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 certificates.

## **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.

[PKI certificates and requirements](#) includes guidance on setting up a cluster to use an external CA.

## **Check certificate expiration**

You can use the `check-expiration` subcommand to check when certificates expire:

```
kubeadm certs check-expiration
```

The output is similar to this:

CERTIFICATE	EXPIRES	RESIDUAL
TIME	CERTIFICATE AUTHORITY	EXTERNALLY MANAGED
admin.conf	Dec 30, 2020 23:36 UTC	
364d		no
apiserver	Dec 30, 2020 23:36 UTC	
364d		no
apiserver-etcd-client	Dec 30, 2020 23:36 UTC	
364d		no
etcd-ca		

<i>apiserver-kubelet-client</i>		Dec 30, 2020 23:36 UTC	
<i>364d ca</i>			no
<i>controller-manager.conf</i>		Dec 30, 2020 23:36 UTC	
<i>364d</i>			no
<i>etcd-healthcheck-client</i>		Dec 30, 2020 23:36 UTC	
<i>364d etcd-ca</i>			no
<i>etcd-peer</i>		Dec 30, 2020 23:36 UTC	
<i>364d etcd-ca</i>			no
<i>etcd-server</i>		Dec 30, 2020 23:36 UTC	
<i>364d etcd-ca</i>			no
<i>front-proxy-client</i>		Dec 30, 2020 23:36 UTC	
<i>364d front-proxy-ca</i>			no
<i>scheduler.conf</i>		Dec 30, 2020 23:36 UTC	
<i>364d</i>			no
 <i>CERTIFICATE AUTHORITY EXTERNALLY MANAGED</i>			
<i>ca</i>		Dec 28, 2029 23:36 UTC	9y
<i>no</i>			
<i>etcd-ca</i>		Dec 28, 2029 23:36 UTC	9y
<i>no</i>			
<i>front-proxy-ca</i>		Dec 28, 2029 23:36 UTC	9y
<i>no</i>			

The command shows expiration/residual time for the client certificates in the `/etc/kubernetes/pki` folder and for the client certificate embedded in the KUBECONFIG files used by kubeadm (`admin.conf`, `controller-manager.conf` and `scheduler.conf`).

Additionally, kubeadm informs the user if the certificate is externally managed; in this case, the user should take care of managing certificate renewal manually/using other tools.

**Warning:** kubeadm cannot manage certificates signed by an external CA.

**Note:** `kubelet.conf` is not included in the list above because kubeadm configures kubelet for automatic certificate renewal.

### Warning:

On nodes created with `kubeadm init`, prior to kubeadm version 1.17, there is a [bug](#) where you manually have to modify the contents of `kubelet.conf`. After `kubeadm init` finishes, you should update `kubelet.conf` to point to the rotated kubelet client certificates, by replacing `client-certificate-data` and `client-key-data` with:

```
client-certificate: /var/lib/kubelet/pki/kubelet-client-current.pem
client-key: /var/lib/kubelet/pki/kubelet-client-current.pem
```

## Automatic certificate renewal

`kubeadm` renews all the certificates during control plane [upgrade](#).

This feature is designed for addressing the simplest use cases; if you don't have specific requirements on certificate renewal and perform Kubernetes version upgrades regularly (less than 1 year in between each upgrade), `kubeadm` will take care of keeping your cluster up to date and reasonably secure.

**Note:** It is a best practice to upgrade your cluster frequently in order to stay secure.

If you have more complex requirements for certificate renewal, you can opt out from the default behavior by passing `--certificate-renewal=false` to `kubeadm upgrade apply` or to `kubeadm upgrade node`.

**Warning:** Prior to `kubeadm` version 1.17 there is a [bug](#) where the default value for `--certificate-renewal` is `false` for the `kubeadm upgrade node` command. In that case, you should explicitly set `--certificate-renewal=true`.

## Manual certificate renewal

You can renew your certificates manually at any time with the `kubeadm certs renew` command.

This command performs the renewal using CA (or front-proxy-CA) certificate and key stored in `/etc/kubernetes/pki`.

**Warning:** If you are running an HA cluster, this command needs to be executed on all the control-plane nodes.

**Note:** `certs renew` uses the existing certificates as the authoritative source for attributes (Common Name, Organization, SAN, etc.) instead of the `kubeadm-config ConfigMap`. It is strongly recommended to keep them both in sync.

`kubeadm certs renew` provides the following options:

The Kubernetes certificates normally reach their expiration date after one year.

- `--csr-only` can be used to renew certificates with an external CA by generating certificate signing requests (without actually renewing certificates in place); see next paragraph for more information.
- It's also possible to renew a single certificate instead of all.

# **Renew certificates with the Kubernetes certificates API**

This section provide more details about how to execute manual certificate renewal using the Kubernetes certificates API.

**Caution:** These are advanced topics for users who need to integrate their organization's certificate infrastructure into a kubeadm-built cluster. If the default kubeadm configuration satisfies your needs, you should let kubeadm manage certificates instead.

## **Set up a signer**

The Kubernetes Certificate Authority does not work out of the box. You can configure an external signer such as [cert-manager](#), or you can use the built-in signer.

The built-in signer is part of [kube-controller-manager](#).

To activate the built-in signer, you must pass the `--cluster-signing-cert-file` and `--cluster-signing-key-file` flags.

If you're creating a new cluster, you can use a kubeadm [configuration file](#):

```
apiVersion: kubeconfig.k8s.io/v1beta2
kind: ClusterConfiguration
controllerManager:
  extraArgs:
    cluster-signing-cert-file: /etc/kubernetes/pki/ca.crt
    cluster-signing-key-file: /etc/kubernetes/pki/ca.key
```

## **Create certificate signing requests (CSR)**

You can create the certificate signing requests for the Kubernetes certificates API with `kubeadm certs renew --use-api`.

If you set up an external signer such as [cert-manager](#), certificate signing requests (CSRs) are automatically approved. Otherwise, you must manually approve certificates with the [kubectl certificate](#) command. The following kubeadm command outputs the name of the certificate to approve, then blocks and waits for approval to occur:

```
sudo kubeadm certs renew apiserver --use-api &
```

The output is similar to this:

```
[1] 2890
[certs] certificate request "kubeadm-cert-kube-apiserver-ld526"
created
```

## **Approve certificate signing requests (CSR)**

If you set up an external signer, certificate signing requests (CSRs) are automatically approved.

Otherwise, you must manually approve certificates with the [`kubectl certificate`](#) command. e.g.

```
kubectl certificate approve kubeadm-cert-kube-apiserver-ld526
```

The output is similar to this:

```
certificatesigningrequest.certificates.k8s.io/kubeadm-cert-kube-apiserver-ld526 approved
```

You can view a list of pending certificates with `kubectl get csr`.

## **Renew certificates with external CA**

This section provide more details about how to execute manual certificate renewal using an external CA.

To better integrate with external CAs, `kubeadm` can also produce certificate signing requests (CSRs). A CSR represents a request to a CA for a signed certificate for a client. In `kubeadm` terms, any certificate that would normally be signed by an on-disk CA can be produced as a CSR instead. A CA, however, cannot be produced as a CSR.

### **Create certificate signing requests (CSR)**

You can create certificate signing requests with `kubeadm certs renew --csr-only`.

Both the CSR and the accompanying private key are given in the output. You can pass in a directory with `--csr-dir` to output the CSRs to the specified location. If `--csr-dir` is not specified, the default certificate directory (`/etc/kubernetes/pki`) is used.

Certificates can be renewed with `kubeadm certs renew --csr-only`. As with `kubeadm init`, an output directory can be specified with the `--csr-dir` flag.

A CSR contains a certificate's name, domains, and IPs, but it does not specify usages. It is the responsibility of the CA to specify [the correct cert usages](#) when issuing a certificate.

- In `openssl` this is done with the [`openssl ca command`](#).
- In `cfls` you specify [`usages in the config file`](#).

After a certificate is signed using your preferred method, the certificate and the private key must be copied to the PKI directory (by default `/etc/kubernetes/pki`).

# Certificate authority (CA) rotation

Kubeadm does not support rotation or replacement of CA certificates out of the box.

For more information about manual rotation or replacement of CA, see [manual rotation of CA certificates](#).

## 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](#).

Last modified November 12, 2020 at 9:28 PM PST: [kubeadm: promote the "kubeadm certs" command to GA \(#24410\) \(d0c6d303c\)](#)  
[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Using custom certificates](#)
- [External CA mode](#)
- [Check certificate expiration](#)
- [Automatic certificate renewal](#)
- [Manual certificate renewal](#)
- [Renew certificates with the Kubernetes certificates API](#)
  - [Set up a signer](#)
  - [Create certificate signing requests \(CSR\)](#)
  - [Approve certificate signing requests \(CSR\)](#)
- [Renew certificates with external CA](#)
  - [Create certificate signing requests \(CSR\)](#)
- [Certificate authority \(CA\) rotation](#)

# Upgrading kubeadm clusters

This page explains how to upgrade a Kubernetes cluster created with kubeadm from version 1.19.x to version 1.20.x, and from version 1.20.x to 1.20.y (where  $y > x$ ). Skipping MINOR versions when upgrading is unsupported.

To see information about upgrading clusters created using older versions of kubeadm, please refer to following pages instead:

- [Upgrading a kubeadm cluster from 1.18 to 1.19](#)
- [Upgrading a kubeadm cluster from 1.17 to 1.18](#)
- [Upgrading a kubeadm cluster from 1.16 to 1.17](#)
- [Upgrading a kubeadm cluster from 1.15 to 1.16](#)

The upgrade workflow at high level is the following:

1. Upgrade a primary control plane node.
2. Upgrade additional control plane nodes.
3. Upgrade worker nodes.

## **Before you begin**

- Make sure you read the [release notes](#) carefully.
- The cluster should use a static control plane and etcd pods or external etcd.
- Make sure to back up any important components, such as app-level state stored in a database. `kubeadm upgrade` does not touch your workloads, only components internal to Kubernetes, but backups are always a best practice.
- [Swap must be disabled](#).

## **Additional information**

- [Draining nodes](#) before kubelet MINOR version upgrades is required. In the case of control plane nodes, they could be running CoreDNS Pods or other critical workloads.
- All containers are restarted after upgrade, because the container spec hash value is changed.

## **Determine which version to upgrade to**

Find the latest stable 1.20 version using the OS package manager:

- [Ubuntu, Debian or HypriotOS](#)
- [CentOS, RHEL or Fedora](#)

```
apt update
apt-cache madison kubeadm
# find the latest 1.20 version in the list
# it should look like 1.20.x-00, where x is the latest patch

yum list --showduplicates kubeadm --disableexcludes=kubernetes
# find the latest 1.20 version in the list
# it should look like 1.20.x-0, where x is the latest patch
```

## **Upgrading control plane nodes**

The upgrade procedure on control plane nodes should be executed one node at a time. Pick a control plane node that you wish to upgrade first. It must have the `/etc/kubernetes/admin.conf` file.

## **Call "kubeadm upgrade"**

### **For the first control plane node**

- Upgrade kubeadm:
  - [Ubuntu, Debian or HypriotOS](#)
  - [CentOS, RHEL or Fedora](#)

```
# replace x in 1.20.x-00 with the latest patch version
apt-mark unhold kubeadm && \
apt-get update && apt-get install -y kubeadm=1.20.x-00 && \
apt-mark hold kubeadm

#
# since apt-get version 1.1 you can also use the following method
apt-get update && \
apt-get install -y --allow-change-held-packages kubeadm=1.20.x-00

# replace x in 1.20.x-0 with the latest patch version
yum install -y kubeadm-1.20.x-0 --disableexcludes=kubernetes
```

- Verify that the download works and has the expected version:

```
kubeadm version
```

- Verify the upgrade plan:

```
kubeadm upgrade plan
```

*This command checks that your cluster can be upgraded, and fetches the versions you can upgrade to. It also shows a table with the component config version states.*

**Note:** *kubeadm upgrade* also automatically renews the certificates that it manages on this node. To opt-out of certificate renewal the flag `--certificate-renewal=false` can be used. For more information see the [certificate management guide](#).

**Note:** If *kubeadm upgrade plan* shows any component configs that require manual upgrade, users must provide a config file with replacement configs to *kubeadm upgrade apply* via the `--config` command line flag. Failing to do so will cause *kubeadm upgrade apply* to exit with an error and not perform an upgrade.

- Choose a version to upgrade to, and run the appropriate command. For example:

```
# replace x with the patch version you picked for this
upgrade
sudo kubeadm upgrade apply v1.20.x
```

Once the command finishes you should see:

```
[upgrade/successful] SUCCESS! Your cluster was upgraded to  
"v1.20.x". Enjoy!
```

```
[upgrade/kubelet] Now that your control plane is upgraded,  
please proceed with upgrading your kubelets if you haven't  
already done so.
```

- Manually upgrade your CNI provider plugin.

Your Container Network Interface (CNI) provider may have its own upgrade instructions to follow. Check the [addons](#) page to find your CNI provider and see whether additional upgrade steps are required.

This step is not required on additional control plane nodes if the CNI provider runs as a DaemonSet.

### For the other control plane nodes

Same as the first control plane node but use:

```
sudo kubeadm upgrade node
```

instead of:

```
sudo kubeadm upgrade apply
```

Also calling `kubeadm upgrade plan` and upgrading the CNI provider plugin is no longer needed.

### Drain the node

- Prepare the node for maintenance by marking it unschedulable and evicting the workloads:

```
# replace <node-to-drain> with the name of your node you are  
draining  
kubectl drain <node-to-drain> --ignore-daemonsets
```

### Upgrade kubelet and kubectl

- Upgrade the kubelet and kubectl
- [Ubuntu, Debian or HypriotOS](#)
- [CentOS, RHEL or Fedora](#)

```
# replace x in 1.20.x-00 with the latest patch version  
apt-mark unhold kubelet kubectl && \  
apt-get update && apt-get install -y kubelet=1.20.x-00  
kubectl=1.20.x-00 && \  
apt-mark hold kubelet kubectl  
-  
# since apt-get version 1.1 you can also use the following  
method
```

```
apt-get update && \
apt-get install -y --allow-change-held-packages
kubelet=1.20.x-00 kubectl=1.20.x-00
```

```
# replace x in 1.20.x-0 with the latest patch version
yum install -y kubelet-1.20.x-0 kubectl-1.20.x-0 --
disableexcludes=kubernetes
```

- Restart the kubelet:

```
sudo systemctl daemon-reload
sudo systemctl restart kubelet
```

## **Uncordon the node**

- Bring the node back online by marking it schedulable:

```
# replace <node-to-drain> with the name of your node
kubectl uncordon <node-to-drain>
```

## **Upgrade worker nodes**

The upgrade procedure on worker nodes should be executed one node at a time or few nodes at a time, without compromising the minimum required capacity for running your workloads.

### **Upgrade kubeadm**

- Upgrade kubeadm:
  - [Ubuntu, Debian or HypriotOS](#)
  - [CentOS, RHEL or Fedora](#)

```
# replace x in 1.20.x-00 with the latest patch version
apt-mark unhold kubeadm && \
apt-get update && apt-get install -y kubeadm=1.20.x-00 && \
apt-mark hold kubeadm
-
# since apt-get version 1.1 you can also use the following method
apt-get update && \
apt-get install -y --allow-change-held-packages kubeadm=1.20.x-00
#
# replace x in 1.20.x-0 with the latest patch version
yum install -y kubeadm-1.20.x-0 --disableexcludes=kubernetes
```

## **Drain the node**

- Prepare the node for maintenance by marking it unschedulable and evicting the workloads:

```
# replace <node-to-drain> with the name of your node you are  
draining  
kubectl drain <node-to-drain> --ignore-daemonsets
```

## Call "kubeadm upgrade"

- For worker nodes this upgrades the local kubelet configuration:

```
sudo kubeadm upgrade node
```

## Upgrade kubelet and kubectl

- Upgrade the kubelet and kubectl:
  - [Ubuntu, Debian or HypriotOS](#)
  - [CentOS, RHEL or Fedora](#)

```
# replace x in 1.20.x-00 with the latest patch version  
apt-mark unhold kubelet kubectl && \  
apt-get update && apt-get install -y kubelet=1.20.x-00  
kubectl=1.20.x-00 && \  
apt-mark hold kubelet kubectl  
-  
# since apt-get version 1.1 you can also use the following method  
apt-get update && \  
apt-get install -y --allow-change-held-packages  
kubelet=1.20.x-00 kubectl=1.20.x-00
```

```
# replace x in 1.20.x-0 with the latest patch version  
yum install -y kubelet-1.20.x-0 kubectl-1.20.x-0 --  
disableexcludes=kubernetes
```

- Restart the kubelet:

```
sudo systemctl daemon-reload  
sudo systemctl restart kubelet
```

## Uncordon the node

- Bring the node back online by marking it schedulable:

```
# replace <node-to-drain> with the name of your node  
kubectl uncordon <node-to-drain>
```

## Verify the status of the cluster

After the kubelet is upgraded on all nodes verify that all nodes are available again by running the following command from anywhere kubectl can access the cluster:

```
kubectl get nodes
```

The *STATUS* column should show *Ready* for all your nodes, and the version number should be updated.

## **Recovering from a failure state**

If `kubeadm upgrade` fails and does not roll back, for example because of an unexpected shutdown during execution, you can run `kubeadm upgrade` again. This command is idempotent and eventually makes sure that the actual state is the desired state you declare.

To recover from a bad state, you can also run `kubeadm upgrade apply --force` without changing the version that your cluster is running.

During upgrade `kubeadm` writes the following backup folders under `/etc/kubernetes/tmp`:

- `kubeadm-backup-etcd-<date>-<time>`
- `kubeadm-backup-manifests-<date>-<time>`

`kubeadm-backup-etcd` contains a backup of the local etcd member data for this control plane Node. In case of an etcd upgrade failure and if the automatic rollback does not work, the contents of this folder can be manually restored in `/var/lib/etcd`. In case external etcd is used this backup folder will be empty.

`kubeadm-backup-manifests` contains a backup of the static Pod manifest files for this control plane Node. In case of a upgrade failure and if the automatic rollback does not work, the contents of this folder can be manually restored in `/etc/kubernetes/manifests`. If for some reason there is no difference between a pre-upgrade and post-upgrade manifest file for a certain component, a backup file for it will not be written.

## **How it works**

`kubeadm upgrade apply` does the following:

- Checks that your cluster is in an upgradeable state:
  - The API server is reachable
  - All nodes are in the Ready state
  - The control plane is healthy
- Enforces the version skew policies.
- Makes sure the control plane images are available or available to pull to the machine.
- Generates replacements and/or uses user supplied overwrites if component configs require version upgrades.
- Upgrades the control plane components or rollbacks if any of them fails to come up.
- Applies the new `kube-dns` and `kube-proxy` manifests and makes sure that all necessary RBAC rules are created.
- Creates new certificate and key files of the API server and backs up old files if they're about to expire in 180 days.

`kubeadm upgrade node` does the following on additional control plane nodes:

- Fetches the `kubeadm ClusterConfiguration` from the cluster.
- Optionally backups the `kube-apiserver` certificate.
- Upgrades the static Pod manifests for the control plane components.
- Upgrades the `kubelet` configuration for this node.

`kubeadm upgrade node` does the following on worker nodes:

- Fetches the `kubeadm ClusterConfiguration` from the cluster.
- Upgrades the `kubelet` configuration for this node.

## 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](#).

Last modified November 05, 2020 at 7:56 PM PST: [kubeadm: upgrade the upgrade documentation for 1.20 \(b675adf79\)](#)

[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
  - [Additional information](#)
- [Determine which version to upgrade to](#)
- [Upgrading control plane nodes](#)
  - [Call "kubeadm upgrade"](#)
  - [Drain the node](#)
  - [Upgrade kubelet and kubectl](#)
  - [Uncordon the node](#)
- [Upgrade worker nodes](#)
  - [Upgrade kubeadm](#)
  - [Drain the node](#)
  - [Call "kubeadm upgrade"](#)
  - [Upgrade kubelet and kubectl](#)
  - [Uncordon the node](#)
- [Verify the status of the cluster](#)
- [Recovering from a failure state](#)
- [How it works](#)

## Adding Windows nodes

**FEATURE STATE:** Kubernetes v1.18 [beta]

You can use Kubernetes to run a mixture of Linux and Windows nodes, so you can mix Pods that run on Linux on with Pods that run on Windows. This page shows how to register Windows nodes to your cluster.

## **Before you begin**

Your Kubernetes server must be at or later than version 1.17. To check the version, enter `kubectl version`.

- Obtain a [Windows Server 2019 license](#) (or higher) in order to configure the Windows node that hosts Windows containers. If you are using VXLAN/Overlay networking you must have also have [KB4489899](#) installed.
- A Linux-based Kubernetes kubeadm cluster in which you have access to the control plane (see [Creating a single control-plane cluster with kubeadm](#)).

## **Objectives**

- Register a Windows node to the cluster
- Configure networking so Pods and Services on Linux and Windows can communicate with each other

## **Getting Started: Adding a Windows Node to Your Cluster**

### **Networking Configuration**

Once you have a Linux-based Kubernetes control-plane node you are ready to choose a networking solution. This guide illustrates using Flannel in VXLAN mode for simplicity.

#### **Configuring Flannel**

##### **1. Prepare Kubernetes control plane for Flannel**

Some minor preparation is recommended on the Kubernetes control plane in our cluster. It is recommended to enable bridged IPv4 traffic to iptables chains when using Flannel. The following command must be run on all Linux nodes:

```
sudo sysctl net.bridge.bridge-nf-call-iptables=1
```

2. Download & configure Flannel for Linux

Download the most recent Flannel manifest:

```
 wget https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml
```

Modify the `net-conf.json` section of the flannel manifest in order to set the VNI to 4096 and the Port to 4789. It should look as follows:

```
net-conf.json: |
  {
    "Network": "10.244.0.0/16",
    "Backend": {
      "Type": "vxlan",
      "VNI": 4096,
      "Port": 4789
    }
  }
```

**Note:** The VNI must be set to 4096 and port 4789 for Flannel on Linux to interoperate with Flannel on Windows. See the [VXLAN documentation](#) for an explanation of these fields.

**Note:** To use L2Bridge/Host-gateway mode instead change the value of Type to "host-gw" and omit VNI and Port.

3. Apply the Flannel manifest and validate

Let's apply the Flannel configuration:

```
kubectl apply -f kube-flannel.yml
```

After a few minutes, you should see all the pods as running if the Flannel pod network was deployed.

```
kubectl get pods -n kube-system
```

The output should include the Linux flannel DaemonSet as running:

NAMESPACE	NAME	READY	STATUS	RESTARTS	AGE
...					
kube-system	kube-flannel-ds-54954	1/1	Running	0	1m

4. Add Windows Flannel and kube-proxy DaemonSets

Now you can add Windows-compatible versions of Flannel and kube-proxy. In order to ensure that you get a compatible version of kube-proxy, you'll need to substitute the tag of the image. The following example shows usage for Kubernetes v1.20.0, but you should adjust the version for your own deployment.

```
curl -L https://github.com/kubernetes-sigs/sig-windows-tools/releases/latest/download/kube-proxy.yml | sed 's/VERSION/v1.20.0/g' | kubectl apply -f -
kubectl apply -f https://github.com/kubernetes-sigs/sig-windows-tools/releases/latest/download/flannel-overlay.yml
```

**Note:** If you're using host-gateway use <https://github.com/kubernetes-sigs/sig-windows-tools/releases/latest/download/flannel-host-gw.yml> instead

### Note:

If you're using a different interface rather than Ethernet (i.e. "Ethernet0 2") on the Windows nodes, you have to modify the line:

```
wins cli process run --path /k/flannel/setup.exe --
args "--mode=overlay --interface=Ethernet"
```

in the flannel-host-gw.yml or flannel-overlay.yml file and specify your interface accordingly.

### # Example

```
curl -L https://github.com/kubernetes-sigs/sig-windows-tools/releases/latest/download/flannel-
overlay.yml | sed 's/Ethernet/Ethernet0 2/g' |
kubectl apply -f -
```

## Joining a Windows worker node

**Note:** All code snippets in Windows sections are to be run in a PowerShell environment with elevated permissions (Administrator) on the Windows worker node.

- [Docker EE](#)
- [CRI-containerD](#)

### Install Docker EE

Install the Containers feature

```
Install-WindowsFeature -Name containers
```

Install Docker Instructions to do so are available at [Install Docker Engine - Enterprise on Windows Servers](#).

## **Install wins, kubelet, and kubeadm**

```
curl.exe -L0 https://github.com/kubernetes-sigs/sig-windows-tools/releases/latest/download/PrepareNode.ps1  
.\\PrepareNode.ps1 -KubernetesVersion v1.20.0
```

### **Run kubeadm to join the node**

*Use the command that was given to you when you ran kubeadm init on a control plane host. If you no longer have this command, or the token has expired, you can run kubeadm token create --print-join-command (on a control plane host) to generate a new token and join command.*

## **Install containerD**

```
curl.exe -L0 https://github.com/kubernetes-sigs/sig-windows-tools/releases/latest/download/Install-Containerd.ps1  
.\\Install-Containerd.ps1
```

### **Note:**

*To install a specific version of containerD specify the version with -ContainerDVersion.*

```
# Example  
.\\Install-Containerd.ps1 -ContainerDVersion v1.4.1
```

### **Note:**

*If you're using a different interface rather than Ethernet (i.e. "Ethernet0 2") on the Windows nodes, specify the name with -netAdapterName.*

```
# Example  
.\\Install-Containerd.ps1 -netAdapterName "Ethernet0 2"
```

## **Install wins, kubelet, and kubeadm**

```
curl.exe -L0 https://github.com/kubernetes-sigs/sig-windows-tools/releases/latest/download/PrepareNode.ps1  
.\\PrepareNode.ps1 -KubernetesVersion v1.20.0 -ContainerRuntime containerD
```

### **Run kubeadm to join the node**

*Use the command that was given to you when you ran kubeadm init on a control plane host. If you no longer have this command, or the token has expired, you can run kubeadm token create --print-join-command (on a control plane host) to generate a new token and join command.*

**Note:** If using **CRI-containerD** add `--cri-socket "npipe:///./pipe/containerd-containerd"` to the `kubeadm` call

## Verifying your installation

You should now be able to view the Windows node in your cluster by running:

```
kubectl get nodes -o wide
```

If your new node is in the `NotReady` state it is likely because the flannel image is still downloading. You can check the progress as before by checking on the flannel pods in the `kube-system` namespace:

```
kubectl -n kube-system get pods -l app=flannel
```

Once the flannel Pod is running, your node should enter the `Ready` state and then be available to handle workloads.

## What's next

- [Upgrading Windows kubeadm nodes](#)

## 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](#).

Last modified November 20, 2020 at 7:10 AM PST: [doc updates for graduating Windows + containerd support to stable \(#24862\) \(b41ee2572\)](#)  
[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Objectives](#)
- [Getting Started: Adding a Windows Node to Your Cluster](#)
  - [Networking Configuration](#)
  - [Joining a Windows worker node](#)
  - [Verifying your installation](#)

- [What's next](#)

# Upgrading Windows nodes

**FEATURE STATE:** Kubernetes v1.18 [beta]

This page explains how to upgrade a Windows node [created with kubeadm](#).

## Before you begin

You need to have a Kubernetes cluster, and the `kubectl` command-line tool must be configured to communicate with your cluster. If you do not already have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:

- [Katacoda](#)
- [Play with Kubernetes](#)

Your Kubernetes server must be at or later than version 1.17. To check the version, enter `kubectl version`.

- Familiarize yourself with [the process for upgrading the rest of your kubeadm cluster](#). You will want to upgrade the control plane nodes before upgrading your Windows nodes.

# Upgrading worker nodes

## Upgrade kubeadm

1. From the Windows node, upgrade kubeadm:

```
# replace v1.20.0 with your desired version
curl.exe -Lo C:\k\kubeadm.exe https://dl.k8s.io/v1.20.0/bin/
windows/amd64/kubeadm.exe
```

## Drain the node

1. From a machine with access to the Kubernetes API, prepare the node for maintenance by marking it unschedulable and evicting the workloads:

```
# replace <node-to-drain> with the name of your node you are
draining
kubectl drain <node-to-drain> --ignore-daemonsets
```

You should see output similar to this:

```
node/ip-172-31-85-18 cordoned  
node/ip-172-31-85-18 drained
```

## Upgrade the kubelet configuration

1. From the Windows node, call the following command to sync new kubelet configuration:

```
kubeadm upgrade node
```

## Upgrade kubelet

1. From the Windows node, upgrade and restart the kubelet:

```
stop-service kubelet  
curl.exe -Lo C:\k\kubelet.exe https://dl.k8s.io/v1.20.0/bin/  
windows/amd64/kubelet.exe  
restart-service kubelet
```

## Uncordon the node

1. From a machine with access to the Kubernetes API, bring the node back online by marking it schedulable:

```
# replace <node-to-drain> with the name of your node  
kubectl uncordon <node-to-drain>
```

## Upgrade kube-proxy

1. From a machine with access to the Kubernetes API, run the following, again replacing v1.20.0 with your desired version:

```
curl -L https://github.com/kubernetes-sigs/sig-windows-tools/  
releases/latest/download/kube-proxy.yml | sed 's/VERSION/  
v1.20.0/g' | kubectl apply -f -
```

## 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](#).

Last modified May 30, 2020 at 3:10 PM PST: [add en pages \(ecc27bbbe\)](#)  
[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Upgrading worker nodes](#)
  - [Upgrade kubeadm](#)
  - [Drain the node](#)
  - [Upgrade the kubelet configuration](#)
  - [Upgrade kubelet](#)
  - [Uncordon the node](#)
  - [Upgrade kube-proxy](#)

# Manage Memory, CPU, and API Resources

---

[Configure Default Memory Requests and Limits for a Namespace](#)

[Configure Default CPU Requests and Limits for a Namespace](#)

[Configure Minimum and Maximum Memory Constraints for a Namespace](#)

[Configure Minimum and Maximum CPU Constraints for a Namespace](#)

[Configure Memory and CPU Quotas for a Namespace](#)

[Configure a Pod Quota for a Namespace](#)

## Configure Default Memory Requests and Limits for a Namespace

This page shows how to configure default memory requests and limits for a namespace. If a Container is created in a namespace that has a default memory limit, and the Container does not specify its own memory limit, then the Container is assigned the default memory limit. Kubernetes assigns a default memory request under certain conditions that are explained later in this topic.

### Before you begin

You need to have a Kubernetes cluster, and the `kubectl` command-line tool must be configured to communicate with your cluster. If you do not already have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:

- [Katacoda](#)
- [Play with Kubernetes](#)

To check the version, enter `kubectl version`.

Each node in your cluster must have at least 2 GiB of memory.

## Create a namespace

Create a namespace so that the resources you create in this exercise are isolated from the rest of your cluster.

```
kubectl create namespace default-mem-example
```

## Create a LimitRange and a Pod

Here's the configuration file for a `LimitRange` object. The configuration specifies a default memory request and a default memory limit.

[admin/resource/memory-defaults.yaml](#)  


```
apiVersion: v1
kind: LimitRange
metadata:
  name: mem-limit-range
spec:
  limits:
  - default:
    memory: 512Mi
  defaultRequest:
    memory: 256Mi
  type: Container
```

Create the `LimitRange` in the `default-mem-example` namespace:

```
kubectl apply -f https://k8s.io/examples/admin/resource/memory-defaults.yaml --namespace=default-mem-example
```

*Now if a Container is created in the default-mem-example namespace, and the Container does not specify its own values for memory request and memory limit, the Container is given a default memory request of 256 MiB and a default memory limit of 512 MiB.*

*Here's the configuration file for a Pod that has one Container. The Container does not specify a memory request and limit.*

[admin/resource/memory-defaults-pod.yaml](#)  


```
apiVersion: v1
kind: Pod
metadata:
  name: default-mem-demo
spec:
  containers:
    - name: default-mem-demo-ctr
      image: nginx
```

*Create the Pod.*

```
kubectl apply -f https://k8s.io/examples/admin/resource/memory-defaults-pod.yaml --namespace=default-mem-example
```

*View detailed information about the Pod:*

```
kubectl get pod default-mem-demo --output=yaml --namespace=default-mem-example
```

*The output shows that the Pod's Container has a memory request of 256 MiB and a memory limit of 512 MiB. These are the default values specified by the LimitRange.*

```
containers:
- image: nginx
  imagePullPolicy: Always
  name: default-mem-demo-ctr
  resources:
    limits:
      memory: 512Mi
    requests:
      memory: 256Mi
```

*Delete your Pod:*

```
kubectl delete pod default-mem-demo --namespace=default-mem-example
```

## What if you specify a Container's limit, but not its request?

Here's the configuration file for a Pod that has one Container. The Container specifies a memory limit, but not a request:

[admin/resource/memory-defaults-pod-2.yaml](#)



```
apiVersion: v1
kind: Pod
metadata:
  name: default-mem-demo-2
spec:
  containers:
    - name: default-mem-demo-2-ctr
      image: nginx
      resources:
        limits:
          memory: "1Gi"
```

Create the Pod:

```
kubectl apply -f https://k8s.io/examples/admin/resource/memory-defaults-pod-2.yaml --namespace=default-mem-example
```

View detailed information about the Pod:

```
kubectl get pod default-mem-demo-2 --output=yaml --namespace=default-mem-example
```

The output shows that the Container's memory request is set to match its memory limit. Notice that the Container was not assigned the default memory request value of 256Mi.

```
resources:
  limits:
    memory: 1Gi
  requests:
    memory: 1Gi
```

## **What if you specify a Container's request, but not its limit?**

Here's the configuration file for a Pod that has one Container. The Container specifies a memory request, but not a limit:

[admin/resource/memory-defaults-pod-3.yaml](https://k8s.io/examples/admin/resource/memory-defaults-pod-3.yaml)  


```
apiVersion: v1
kind: Pod
metadata:
  name: default-mem-demo-3
spec:
  containers:
    - name: default-mem-demo-3-ctr
      image: nginx
      resources:
        requests:
          memory: "128Mi"
```

Create the Pod:

```
kubectl apply -f https://k8s.io/examples/admin/resource/memory-defaults-pod-3.yaml --namespace=default-mem-example
```

View the Pod's specification:

```
kubectl get pod default-mem-demo-3 --output=yaml --namespace=default-mem-example
```

The output shows that the Container's memory request is set to the value specified in the Container's configuration file. The Container's memory limit is set to 512Mi, which is the default memory limit for the namespace.

```
resources:
  limits:
    memory: 512Mi
  requests:
    memory: 128Mi
```

# **Motivation for default memory limits and requests**

*If your namespace has a resource quota, it is helpful to have a default value in place for memory limit. Here are two of the restrictions that a resource quota imposes on a namespace:*

- *Every Container that runs in the namespace must have its own memory limit.*
- *The total amount of memory used by all Containers in the namespace must not exceed a specified limit.*

*If a Container does not specify its own memory limit, it is given the default limit, and then it can be allowed to run in a namespace that is restricted by a quota.*

## **Clean up**

*Delete your namespace:*

```
kubectl delete namespace default-mem-example
```

## **What's next**

### **For cluster administrators**

- [Configure Default CPU Requests and Limits for a Namespace](#)
- [Configure Minimum and Maximum Memory Constraints for a Namespace](#)
- [Configure Minimum and Maximum CPU Constraints for a Namespace](#)
- [Configure Memory and CPU Quotas for a Namespace](#)
- [Configure a Pod Quota for a Namespace](#)
- [Configure Quotas for API Objects](#)

## **For app developers**

- [Assign Memory Resources to Containers and Pods](#)
- [Assign CPU Resources to Containers and Pods](#)
- [Configure Quality of Service for Pods](#)

## **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](#).

Last modified July 29, 2020 at 9:40 PM PST: [Fix broken links to pages under /en/docs/tasks/administer-cluster/manage-resources/ \(36d9239fb\)](#)  
[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Create a namespace](#)
- [Create a LimitRange and a Pod](#)
- [What if you specify a Container's limit, but not its request?](#)
- [What if you specify a Container's request, but not its limit?](#)
- [Motivation for default memory limits and requests](#)
- [Clean up](#)
- [What's next](#)
  - [For cluster administrators](#)
  - [For app developers](#)

# **Configure Default CPU Requests and Limits for a Namespace**

This page shows how to configure default CPU requests and limits for a namespace. A Kubernetes cluster can be divided into namespaces. If a Container is created in a namespace that has a default CPU limit, and the Container does not specify its own CPU limit, then the Container is assigned the default CPU limit. Kubernetes assigns a default CPU request under certain conditions that are explained later in this topic.

## **Before you begin**

You need to have a Kubernetes cluster, and the `kubectl` command-line tool must be configured to communicate with your cluster. If you do not already have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:

- [Katacoda](#)
- [Play with Kubernetes](#)

To check the version, enter `kubectl version`.

## Create a namespace

Create a namespace so that the resources you create in this exercise are isolated from the rest of your cluster.

```
kubectl create namespace default-cpu-example
```

## Create a LimitRange and a Pod

Here's the configuration file for a `LimitRange` object. The configuration specifies a default CPU request and a default CPU limit.

[admin/resource/cpu-defaults.yaml](#)  


```
apiVersion: v1
kind: LimitRange
metadata:
  name: cpu-limit-range
spec:
  limits:
  - default:
    cpu: 1
  defaultRequest:
    cpu: 0.5
  type: Container
```

Create the `LimitRange` in the `default-cpu-example` namespace:

```
kubectl apply -f https://k8s.io/examples/admin/resource/cpu-
defaults.yaml --namespace=default-cpu-example
```

Now if a Container is created in the `default-cpu-example` namespace, and the Container does not specify its own values for CPU request and CPU

*limit, the Container is given a default CPU request of 0.5 and a default CPU limit of 1.*

*Here's the configuration file for a Pod that has one Container. The Container does not specify a CPU request and limit.*

[admin/resource/cpu-defaults-pod.yaml](#)



```
apiVersion: v1
kind: Pod
metadata:
  name: default-cpu-demo
spec:
  containers:
    - name: default-cpu-demo-ctr
      image: nginx
```

*Create the Pod.*

```
kubectl apply -f https://k8s.io/examples/admin/resource/cpu-
defaults-pod.yaml --namespace=default-cpu-example
```

*View the Pod's specification:*

```
kubectl get pod default-cpu-demo --output=yaml --namespace=defaul
t-cpu-example
```

*The output shows that the Pod's Container has a CPU request of 500 millicpus and a CPU limit of 1 cpu. These are the default values specified by the LimitRange.*

```
containers:
- image: nginx
  imagePullPolicy: Always
  name: default-cpu-demo-ctr
  resources:
    limits:
      cpu: "1"
    requests:
      cpu: 500m
```

## **What if you specify a Container's limit, but not its request?**

Here's the configuration file for a Pod that has one Container. The Container specifies a CPU limit, but not a request:

[admin/resource/cpu-defaults-pod-2.yaml](https://k8s.io/examples/admin/resource/cpu-defaults-pod-2.yaml)  


```
apiVersion: v1
kind: Pod
metadata:
  name: default-cpu-demo-2
spec:
  containers:
    - name: default-cpu-demo-2-ctr
      image: nginx
      resources:
        limits:
          cpu: "1"
```

Create the Pod:

```
kubectl apply -f https://k8s.io/examples/admin/resource/cpu-defaults-pod-2.yaml --namespace=default-cpu-example
```

View the Pod specification:

```
kubectl get pod default-cpu-demo-2 --output=yaml --
namespace=default-cpu-example
```

The output shows that the Container's CPU request is set to match its CPU limit. Notice that the Container was not assigned the default CPU request value of 0.5 cpu.

```
resources:
  limits:
    cpu: "1"
  requests:
    cpu: "1"
```

## **What if you specify a Container's request, but not its limit?**

Here's the configuration file for a Pod that has one Container. The Container specifies a CPU request, but not a limit:

[admin/resource/cpu-defaults-pod-3.yaml](https://k8s.io/examples/admin/resource/cpu-defaults-pod-3.yaml)  


```
apiVersion: v1
kind: Pod
metadata:
  name: default-cpu-demo-3
spec:
  containers:
    - name: default-cpu-demo-3-ctr
      image: nginx
      resources:
        requests:
          cpu: "0.75"
```

Create the Pod:

```
kubectl apply -f https://k8s.io/examples/admin/resource/cpu-defaults-pod-3.yaml --namespace=default-cpu-example
```

View the Pod specification:

```
kubectl get pod default-cpu-demo-3 --output=yaml --
namespace=default-cpu-example
```

The output shows that the Container's CPU request is set to the value specified in the Container's configuration file. The Container's CPU limit is set to 1 cpu, which is the default CPU limit for the namespace.

```
resources:
  limits:
    cpu: "1"
  requests:
    cpu: 750m
```

# **Motivation for default CPU limits and requests**

*If your namespace has a [resource quota](#), it is helpful to have a default value in place for CPU limit. Here are two of the restrictions that a resource quota imposes on a namespace:*

- *Every Container that runs in the namespace must have its own CPU limit.*
- *The total amount of CPU used by all Containers in the namespace must not exceed a specified limit.*

*If a Container does not specify its own CPU limit, it is given the default limit, and then it can be allowed to run in a namespace that is restricted by a quota.*

## **Clean up**

*Delete your namespace:*

```
kubectl delete namespace default-cpu-example
```

## **What's next**

### **For cluster administrators**

- [Configure Default Memory Requests and Limits for a Namespace](#)
- [Configure Minimum and Maximum Memory Constraints for a Namespace](#)
- [Configure Minimum and Maximum CPU Constraints for a Namespace](#)
- [Configure Memory and CPU Quotas for a Namespace](#)
- [Configure a Pod Quota for a Namespace](#)
- [Configure Quotas for API Objects](#)

## **For app developers**

- [Assign Memory Resources to Containers and Pods](#)
- [Assign CPU Resources to Containers and Pods](#)
- [Configure Quality of Service for Pods](#)

## **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](#).

Last modified July 29, 2020 at 9:40 PM PST: [Fix broken links to pages under /en/docs/tasks/administer-cluster/manage-resources/ \(36d9239fb\)](#)  
[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Create a namespace](#)
- [Create a LimitRange and a Pod](#)
- [What if you specify a Container's limit, but not its request?](#)
- [What if you specify a Container's request, but not its limit?](#)
- [Motivation for default CPU limits and requests](#)
- [Clean up](#)
- [What's next](#)
  - [For cluster administrators](#)
  - [For app developers](#)

# **Configure Minimum and Maximum Memory Constraints for a Namespace**

This page shows how to set minimum and maximum values for memory used by Containers running in a namespace. You specify minimum and maximum memory values in a [LimitRange](#) object. If a Pod does not meet the constraints imposed by the LimitRange, it cannot be created in the namespace.

## **Before you begin**

You need to have a Kubernetes cluster, and the `kubectl` command-line tool must be configured to communicate with your cluster. If you do not already have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:

- [Katacoda](#)
- [Play with Kubernetes](#)

To check the version, enter `kubectl version`.

Each node in your cluster must have at least 1 GiB of memory.

## Create a namespace

Create a namespace so that the resources you create in this exercise are isolated from the rest of your cluster.

```
kubectl create namespace constraints-mem-example
```

## Create a LimitRange and a Pod

Here's the configuration file for a LimitRange:

[admin/resource/memory-constraints.yaml](#)  


```
apiVersion: v1
kind: LimitRange
metadata:
  name: mem-min-max-demo-lr
spec:
  limits:
  - max:
      memory: 1Gi
    min:
      memory: 500Mi
    type: Container
```

Create the LimitRange:

```
kubectl apply -f https://k8s.io/examples/admin/resource/memory-
constraints.yaml --namespace=constraints-mem-example
```

View detailed information about the LimitRange:

```
kubectl get limitrange mem-min-max-demo-lr --namespace=constraints-mem-example --output=yaml
```

The output shows the minimum and maximum memory constraints as expected. But notice that even though you didn't specify default values in the configuration file for the LimitRange, they were created automatically.

```
limits:  
- default:  
  memory: 1Gi  
defaultRequest:  
  memory: 1Gi  
max:  
  memory: 1Gi  
min:  
  memory: 500Mi  
type: Container
```

Now whenever a Container is created in the constraints-mem-example namespace, Kubernetes performs these steps:

- If the Container does not specify its own memory request and limit, assign the default memory request and limit to the Container.
- Verify that the Container has a memory request that is greater than or equal to 500 MiB.
- Verify that the Container has a memory limit that is less than or equal to 1 GiB.

Here's the configuration file for a Pod that has one Container. The Container manifest specifies a memory request of 600 MiB and a memory limit of 800 MiB. These satisfy the minimum and maximum memory constraints imposed by the LimitRange.

[admin/resource/memory-constraints-pod.yaml](#)  


```
apiVersion: v1  
kind: Pod  
metadata:  
  name: constraints-mem-demo  
spec:  
  containers:  
    - name: constraints-mem-demo-ctr  
      image: nginx  
      resources:
```

```
limits:  
  memory: "800Mi"  
requests:  
  memory: "600Mi"
```

*Create the Pod:*

```
kubectl apply -f https://k8s.io/examples/admin/resource/memory-  
constraints-pod.yaml --namespace=constraints-mem-example
```

*Verify that the Pod's Container is running:*

```
kubectl get pod constraints-mem-demo --namespace=constraints-mem-  
example
```

*View detailed information about the Pod:*

```
kubectl get pod constraints-mem-demo --output=yaml --namespace=co  
nstraints-mem-example
```

*The output shows that the Container has a memory request of 600 MiB and a memory limit of 800 MiB. These satisfy the constraints imposed by the LimitRange.*

```
resources:  
limits:  
  memory: 800Mi  
requests:  
  memory: 600Mi
```

*Delete your Pod:*

```
kubectl delete pod constraints-mem-demo --namespace=constraints-  
mem-example
```

## **Attempt to create a Pod that exceeds the maximum memory constraint**

*Here's the configuration file for a Pod that has one Container. The Container specifies a memory request of 800 MiB and a memory limit of 1.5 GiB.*

[admin/resource/memory-constraints-pod-2.yaml](#)



```
apiVersion: v1
kind: Pod
metadata:
  name: constraints-mem-demo-2
spec:
  containers:
    - name: constraints-mem-demo-2-ctr
      image: nginx
      resources:
        limits:
          memory: "1.5Gi"
        requests:
          memory: "800Mi"
```

Attempt to create the Pod:

```
kubectl apply -f https://k8s.io/examples/admin/resource/memory-constraints-pod-2.yaml --namespace=constraints-mem-example
```

The output shows that the Pod does not get created, because the Container specifies a memory limit that is too large:

```
Error from server (Forbidden): error when creating "examples/admin/resource/memory-constraints-pod-2.yaml": pods "constraints-mem-demo-2" is forbidden: maximum memory usage per Container is 1Gi, but limit is 1536Mi.
```

## **Attempt to create a Pod that does not meet the minimum memory request**

Here's the configuration file for a Pod that has one Container. The Container specifies a memory request of 100 MiB and a memory limit of 800 MiB.

[admin/resource/memory-constraints-pod-3.yaml](#)



```
apiVersion: v1
kind: Pod
metadata:
  name: constraints-mem-demo-3
spec:
  containers:
```

```
- name: constraints-mem-demo-3-ctr
  image: nginx
  resources:
    limits:
      memory: "800Mi"
    requests:
      memory: "100Mi"
```

Attempt to create the Pod:

```
kubectl apply -f https://k8s.io/examples/admin/resource/memory-constraints-pod-3.yaml --namespace=constraints-mem-example
```

The output shows that the Pod does not get created, because the Container specifies a memory request that is too small:

```
Error from server (Forbidden): error when creating "examples/admin/resource/memory-constraints-pod-3.yaml": pods "constraints-mem-demo-3" is forbidden: minimum memory usage per Container is 500Mi, but request is 100Mi.
```

## Create a Pod that does not specify any memory request or limit

Here's the configuration file for a Pod that has one Container. The Container does not specify a memory request, and it does not specify a memory limit.

[admin/resource/memory-constraints-pod-4.yaml](https://k8s.io/examples/admin/resource/memory-constraints-pod-4.yaml)  


```
apiVersion: v1
kind: Pod
metadata:
  name: constraints-mem-demo-4
spec:
  containers:
    - name: constraints-mem-demo-4-ctr
      image: nginx
```

Create the Pod:

```
kubectl apply -f https://k8s.io/examples/admin/resource/memory-constraints-pod-4.yaml --namespace=constraints-mem-example
```

*View detailed information about the Pod:*

```
kubectl get pod constraints-mem-demo-4 --namespace=constraints-mem-example --output=yaml
```

*The output shows that the Pod's Container has a memory request of 1 GiB and a memory limit of 1 GiB. How did the Container get those values?*

```
resources:  
  limits:  
    memory: 1Gi  
  requests:  
    memory: 1Gi
```

*Because your Container did not specify its own memory request and limit, it was given the [default memory request and limit](#) from the LimitRange.*

*At this point, your Container might be running or it might not be running. Recall that a prerequisite for this task is that your Nodes have at least 1 GiB of memory. If each of your Nodes has only 1 GiB of memory, then there is not enough allocatable memory on any Node to accommodate a memory request of 1 GiB. If you happen to be using Nodes with 2 GiB of memory, then you probably have enough space to accommodate the 1 GiB request.*

*Delete your Pod:*

```
kubectl delete pod constraints-mem-demo-4 --namespace=constraints-mem-example
```

## **Enforcement of minimum and maximum memory constraints**

*The maximum and minimum memory constraints imposed on a namespace by a LimitRange are enforced only when a Pod is created or updated. If you change the LimitRange, it does not affect Pods that were created previously.*

## **Motivation for minimum and maximum memory constraints**

*As a cluster administrator, you might want to impose restrictions on the amount of memory that Pods can use. For example:*

- *Each Node in a cluster has 2 GB of memory. You do not want to accept any Pod that requests more than 2 GB of memory, because no Node in the cluster can support the request.*
- *A cluster is shared by your production and development departments. You want to allow production workloads to consume up to 8 GB of memory, but you want development workloads to be limited to 512 MB. You create separate namespaces for production and development, and you apply memory constraints to each namespace.*

## **Clean up**

*Delete your namespace:*

```
kubectl delete namespace constraints-mem-example
```

## **What's next**

### **For cluster administrators**

- [Configure Default Memory Requests and Limits for a Namespace](#)
- [Configure Default CPU Requests and Limits for a Namespace](#)
- [Configure Minimum and Maximum CPU Constraints for a Namespace](#)
- [Configure Memory and CPU Quotas for a Namespace](#)
- [Configure a Pod Quota for a Namespace](#)
- [Configure Quotas for API Objects](#)

### **For app developers**

- [Assign Memory Resources to Containers and Pods](#)
- [Assign CPU Resources to Containers and Pods](#)
- [Configure Quality of Service for Pods](#)

# 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](#).

Last modified July 29, 2020 at 9:40 PM PST: [Fix broken links to pages under /en/docs/tasks/administer-cluster/manage-resources/ \(36d9239fb\)](#)

[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Create a namespace](#)
- [Create a LimitRange and a Pod](#)
- [Attempt to create a Pod that exceeds the maximum memory constraint](#)
- [Attempt to create a Pod that does not meet the minimum memory request](#)
- [Create a Pod that does not specify any memory request or limit](#)
- [Enforcement of minimum and maximum memory constraints](#)
- [Motivation for minimum and maximum memory constraints](#)
- [Clean up](#)
- [What's next](#)
  - [For cluster administrators](#)
  - [For app developers](#)

# Configure Minimum and Maximum CPU Constraints for a Namespace

This page shows how to set minimum and maximum values for the CPU resources used by Containers and Pods in a namespace. You specify minimum and maximum CPU values in a [LimitRange](#) object. If a Pod does not meet the constraints imposed by the LimitRange, it cannot be created in the namespace.

## Before you begin

You need to have a Kubernetes cluster, and the `kubectl` command-line tool must be configured to communicate with your cluster. If you do not already have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:

- [Katacoda](#)
- [Play with Kubernetes](#)

To check the version, enter `kubectl version`.

Your cluster must have at least 1 CPU available for use to run the task examples.

## Create a namespace

Create a namespace so that the resources you create in this exercise are isolated from the rest of your cluster.

```
kubectl create namespace constraints-cpu-example
```

## Create a LimitRange and a Pod

Here's the configuration file for a LimitRange:

[admin/resource/cpu-constraints.yaml](#)  


```
apiVersion: v1
kind: LimitRange
metadata:
  name: cpu-min-max-demo-lr
spec:
  limits:
  - max:
      cpu: "800m"
    min:
      cpu: "200m"
  type: Container
```

Create the LimitRange:

```
kubectl apply -f https://k8s.io/examples/admin/resource/cpu-
constraints.yaml --namespace=constraints-cpu-example
```

View detailed information about the LimitRange:

```
kubectl get limitrange cpu-min-max-demo-lr --output=yaml --
namespace=constraints-cpu-example
```

The output shows the minimum and maximum CPU constraints as expected. But notice that even though you didn't specify default values in the configuration file for the LimitRange, they were created automatically.

```
limits:  
- default:  
  cpu: 800m  
  defaultRequest:  
    cpu: 800m  
  max:  
    cpu: 800m  
  min:  
    cpu: 200m  
  type: Container
```

Now whenever a Container is created in the constraints-cpu-example namespace, Kubernetes performs these steps:

- If the Container does not specify its own CPU request and limit, assign the default CPU request and limit to the Container.
- Verify that the Container specifies a CPU request that is greater than or equal to 200 millicpu.
- Verify that the Container specifies a CPU limit that is less than or equal to 800 millicpu.

**Note:** When creating a LimitRange object, you can specify limits on huge-pages or GPUs as well. However, when both default and defaultRequest are specified on these resources, the two values must be the same.

Here's the configuration file for a Pod that has one Container. The Container manifest specifies a CPU request of 500 millicpu and a CPU limit of 800 millicpu. These satisfy the minimum and maximum CPU constraints imposed by the LimitRange.

[admin/resource/cpu-constraints-pod.yaml](#)  


```
apiVersion: v1  
kind: Pod  
metadata:  
  name: constraints-cpu-demo  
spec:  
  containers:  
  - name: constraints-cpu-demo-ctr  
    image: nginx
```

```
resources:  
  limits:  
    cpu: "800m"  
  requests:  
    cpu: "500m"
```

Create the Pod:

```
kubectl apply -f https://k8s.io/examples/admin/resource/cpu-  
constraints-pod.yaml --namespace=constraints-cpu-example
```

Verify that the Pod's Container is running:

```
kubectl get pod constraints-cpu-demo --namespace=constraints-cpu-  
example
```

View detailed information about the Pod:

```
kubectl get pod constraints-cpu-demo --output=yaml --namespace=co  
nstraints-cpu-example
```

The output shows that the Container has a CPU request of 500 millicpu and CPU limit of 800 millicpu. These satisfy the constraints imposed by the LimitRange.

```
resources:  
  limits:  
    cpu: 800m  
  requests:  
    cpu: 500m
```

## Delete the Pod

```
kubectl delete pod constraints-cpu-demo --namespace=constraints-  
cpu-example
```

## Attempt to create a Pod that exceeds the maximum CPU constraint

Here's the configuration file for a Pod that has one Container. The Container specifies a CPU request of 500 millicpu and a cpu limit of 1.5 cpu.

[admin/resource/cpu-constraints-pod-2.yaml](#)



```
apiVersion: v1
kind: Pod
metadata:
  name: constraints-cpu-demo-2
spec:
  containers:
    - name: constraints-cpu-demo-2-ctr
      image: nginx
      resources:
        limits:
          cpu: "1.5"
        requests:
          cpu: "500m"
```

Attempt to create the Pod:

```
kubectl apply -f https://k8s.io/examples/admin/resource/cpu-
constraints-pod-2.yaml --namespace=constraints-cpu-example
```

The output shows that the Pod does not get created, because the Container specifies a CPU limit that is too large:

```
Error from server (Forbidden): error when creating "examples/
admin/resource/cpu-constraints-pod-2.yaml":
pods "constraints-cpu-demo-2" is forbidden: maximum cpu usage
per Container is 800m, but limit is 1500m.
```

## **Attempt to create a Pod that does not meet the minimum CPU request**

Here's the configuration file for a Pod that has one Container. The Container specifies a CPU request of 100 millicpu and a CPU limit of 800 millicpu.

[admin/resource/cpu-constraints-pod-3.yaml](#)



```
apiVersion: v1
kind: Pod
metadata:
  name: constraints-cpu-demo-3
spec:
  containers:
```

```
- name: constraints-cpu-demo-3-ctr
  image: nginx
  resources:
    limits:
      cpu: "800m"
    requests:
      cpu: "100m"
```

Attempt to create the Pod:

```
kubectl apply -f https://k8s.io/examples/admin/resource/cpu-
constraints-pod-3.yaml --namespace=constraints-cpu-example
```

The output shows that the Pod does not get created, because the Container specifies a CPU request that is too small:

```
Error from server (Forbidden): error when creating "examples/
admin/resource/cpu-constraints-pod-3.yaml":
pods "constraints-cpu-demo-3" is forbidden: minimum cpu usage
per Container is 200m, but request is 100m.
```

## Create a Pod that does not specify any CPU request or limit

Here's the configuration file for a Pod that has one Container. The Container does not specify a CPU request, and it does not specify a CPU limit.

[admin/resource/cpu-constraints-pod-4.yaml](#)  


```
apiVersion: v1
kind: Pod
metadata:
  name: constraints-cpu-demo-4
spec:
  containers:
    - name: constraints-cpu-demo-4-ctr
      image: vish/stress
```

Create the Pod:

```
kubectl apply -f https://k8s.io/examples/admin/resource/cpu-
constraints-pod-4.yaml --namespace=constraints-cpu-example
```

*View detailed information about the Pod:*

```
kubectl get pod constraints-cpu-demo-4 --namespace=constraints-cpu-example --output=yaml
```

*The output shows that the Pod's Container has a CPU request of 800 millicpu and a CPU limit of 800 millicpu. How did the Container get those values?*

```
resources:  
  limits:  
    cpu: 800m  
  requests:  
    cpu: 800m
```

*Because your Container did not specify its own CPU request and limit, it was given the [default CPU request and limit](#) from the LimitRange.*

*At this point, your Container might be running or it might not be running. Recall that a prerequisite for this task is that your cluster must have at least 1 CPU available for use. If each of your Nodes has only 1 CPU, then there might not be enough allocatable CPU on any Node to accommodate a request of 800 millicpu. If you happen to be using Nodes with 2 CPU, then you probably have enough CPU to accommodate the 800 millicpu request.*

*Delete your Pod:*

```
kubectl delete pod constraints-cpu-demo-4 --namespace=constraints-cpu-example
```

## ***Enforcement of minimum and maximum CPU constraints***

*The maximum and minimum CPU constraints imposed on a namespace by a LimitRange are enforced only when a Pod is created or updated. If you change the LimitRange, it does not affect Pods that were created previously.*

## ***Motivation for minimum and maximum CPU constraints***

*As a cluster administrator, you might want to impose restrictions on the CPU resources that Pods can use. For example:*

- *Each Node in a cluster has 2 CPU. You do not want to accept any Pod that requests more than 2 CPU, because no Node in the cluster can support the request.*
- *A cluster is shared by your production and development departments. You want to allow production workloads to consume up to 3 CPU, but you want development workloads to be limited to 1 CPU. You create separate namespaces for production and development, and you apply CPU constraints to each namespace.*

## **Clean up**

*Delete your namespace:*

```
kubectl delete namespace constraints-cpu-example
```

## **What's next**

### **For cluster administrators**

- [Configure Default Memory Requests and Limits for a Namespace](#)
- [Configure Default CPU Requests and Limits for a Namespace](#)
- [Configure Minimum and Maximum Memory Constraints for a Namespace](#)
- [Configure Memory and CPU Quotas for a Namespace](#)
- [Configure a Pod Quota for a Namespace](#)
- [Configure Quotas for API Objects](#)

### **For app developers**

- [Assign Memory Resources to Containers and Pods](#)
- [Assign CPU Resources to Containers and Pods](#)
- [Configure Quality of Service for Pods](#)

# 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](#).

Last modified July 29, 2020 at 9:40 PM PST: [Fix broken links to pages under /en/docs/tasks/administer-cluster/manage-resources/ \(36d9239fb\)](#)

[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Create a namespace](#)
- [Create a LimitRange and a Pod](#)
- [Delete the Pod](#)
- [Attempt to create a Pod that exceeds the maximum CPU constraint](#)
- [Attempt to create a Pod that does not meet the minimum CPU request](#)
- [Create a Pod that does not specify any CPU request or limit](#)
- [Enforcement of minimum and maximum CPU constraints](#)
- [Motivation for minimum and maximum CPU constraints](#)
- [Clean up](#)
- [What's next](#)
  - [For cluster administrators](#)
  - [For app developers](#)

# Configure Memory and CPU Quotas for a Namespace

This page shows how to set quotas for the total amount memory and CPU that can be used by all Containers running in a namespace. You specify quotas in a [ResourceQuota](#) object.

## Before you begin

You need to have a Kubernetes cluster, and the `kubectl` command-line tool must be configured to communicate with your cluster. If you do not already have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:

- [Katacoda](#)
- [Play with Kubernetes](#)

To check the version, enter `kubectl version`.

Each node in your cluster must have at least 1 GiB of memory.

## Create a namespace

Create a namespace so that the resources you create in this exercise are isolated from the rest of your cluster.

```
kubectl create namespace quota-mem-cpu-example
```

## Create a ResourceQuota

Here is the configuration file for a ResourceQuota object:

[admin/resource/quota-mem-cpu.yaml](#)  


```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: mem-cpu-demo
spec:
  hard:
    requests.cpu: "1"
    requests.memory: 1Gi
    limits.cpu: "2"
    limits.memory: 2Gi
```

Create the ResourceQuota:

```
kubectl apply -f https://k8s.io/examples/admin/resource/quota-
mem-cpu.yaml --namespace=quota-mem-cpu-example
```

View detailed information about the ResourceQuota:

```
kubectl get resourcequota mem-cpu-demo --namespace=quota-mem-cpu-
example --output=yaml
```

The ResourceQuota places these requirements on the quota-mem-cpu-example namespace:

- Every Container must have a memory request, memory limit, cpu request, and cpu limit.
- The memory request total for all Containers must not exceed 1 GiB.
- The memory limit total for all Containers must not exceed 2 GiB.

- The CPU request total for all Containers must not exceed 1 cpu.
- The CPU limit total for all Containers must not exceed 2 cpu.

## Create a Pod

Here is the configuration file for a Pod:

[admin/resource/quota-mem-cpu-pod.yaml](https://k8s.io/examples/admin/resource/quota-mem-cpu-pod.yaml)  


```
apiVersion: v1
kind: Pod
metadata:
  name: quota-mem-cpu-demo
spec:
  containers:
    - name: quota-mem-cpu-demo-ctr
      image: nginx
      resources:
        limits:
          memory: "800Mi"
          cpu: "800m"
        requests:
          memory: "600Mi"
          cpu: "400m"
```

Create the Pod:

```
kubectl apply -f https://k8s.io/examples/admin/resource/quota-mem-cpu-pod.yaml --namespace=quota-mem-cpu-example
```

Verify that the Pod's Container is running:

```
kubectl get pod quota-mem-cpu-demo --namespace=quota-mem-cpu-example
```

Once again, view detailed information about the ResourceQuota:

```
kubectl get resourcequota mem-cpu-demo --namespace=quota-mem-cpu-example --output=yaml
```

The output shows the quota along with how much of the quota has been used. You can see that the memory and CPU requests and limits for your Pod do not exceed the quota.

```
status:  
  hard:  
    limits.cpu: "2"  
    limits.memory: 2Gi  
    requests.cpu: "1"  
    requests.memory: 1Gi  
  used:  
    limits.cpu: 800m  
    limits.memory: 800Mi  
    requests.cpu: 400m  
    requests.memory: 600Mi
```

## Attempt to create a second Pod

Here is the configuration file for a second Pod:

[admin/resource/quota-mem-cpu-pod-2.yaml](https://k8s.io/examples/admin/resource/quota-mem-cpu-pod-2.yaml)  


```
apiVersion: v1  
kind: Pod  
metadata:  
  name: quota-mem-cpu-demo-2  
spec:  
  containers:  
    - name: quota-mem-cpu-demo-2-ctr  
      image: redis  
      resources:  
        limits:  
          memory: "1Gi"  
          cpu: "800m"  
        requests:  
          memory: "700Mi"  
          cpu: "400m"
```

In the configuration file, you can see that the Pod has a memory request of 700 MiB. Notice that the sum of the used memory request and this new memory request exceeds the memory request quota.  $600 \text{ MiB} + 700 \text{ MiB} > 1 \text{ GiB}$ .

Attempt to create the Pod:

```
kubectl apply -f https://k8s.io/examples/admin/resource/quota-mem-cpu-pod-2.yaml --namespace=quota-mem-cpu-example
```

*The second Pod does not get created. The output shows that creating the second Pod would cause the memory request total to exceed the memory request quota.*

```
Error from server (Forbidden): error when creating "examples/admin/resource/quota-mem-cpu-pod-2.yaml": pods "quota-mem-cpu-demo-2" is forbidden: exceeded quota: mem-cpu-demo, requested: requests.memory=700Mi, used: requests.memory=600Mi, limited: requests.memory=1Gi
```

## **Discussion**

*As you have seen in this exercise, you can use a ResourceQuota to restrict the memory request total for all Containers running in a namespace. You can also restrict the totals for memory limit, cpu request, and cpu limit.*

*If you want to restrict individual Containers, instead of totals for all Containers, use a [LimitRange](#).*

## **Clean up**

*Delete your namespace:*

```
kubectl delete namespace quota-mem-cpu-example
```

## **What's next**

### **For cluster administrators**

- [Configure Default Memory Requests and Limits for a Namespace](#)
- [Configure Default CPU Requests and Limits for a Namespace](#)
- [Configure Minimum and Maximum Memory Constraints for a Namespace](#)
- [Configure Minimum and Maximum CPU Constraints for a Namespace](#)
- [Configure a Pod Quota for a Namespace](#)
- [Configure Quotas for API Objects](#)

## **For app developers**

- [Assign Memory Resources to Containers and Pods](#)
- [Assign CPU Resources to Containers and Pods](#)
- [Configure Quality of Service for Pods](#)

## **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](#).

Last modified July 29, 2020 at 9:40 PM PST: [Fix broken links to pages under /en/docs/tasks/administer-cluster/manage-resources/ \(36d9239fb\)](#)  
[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Create a namespace](#)
- [Create a ResourceQuota](#)
- [Create a Pod](#)
- [Attempt to create a second Pod](#)
- [Discussion](#)
- [Clean up](#)
- [What's next](#)
  - [For cluster administrators](#)
  - [For app developers](#)

# **Configure a Pod Quota for a Namespace**

This page shows how to set a quota for the total number of Pods that can run in a namespace. You specify quotas in a [ResourceQuota](#) object.

## **Before you begin**

You need to have a Kubernetes cluster, and the `kubectl` command-line tool must be configured to communicate with your cluster. If you do not already

have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:

- [Katacoda](#)
- [Play with Kubernetes](#)

To check the version, enter `kubectl version`.

## Create a namespace

Create a namespace so that the resources you create in this exercise are isolated from the rest of your cluster.

```
kubectl create namespace quota-pod-example
```

## Create a ResourceQuota

Here is the configuration file for a ResourceQuota object:

[admin/resource/quota-pod.yaml](#)



```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: pod-demo
spec:
  hard:
    pods: "2"
```

Create the ResourceQuota:

```
kubectl apply -f https://k8s.io/examples/admin/resource/quota-pod.yaml --namespace=quota-pod-example
```

View detailed information about the ResourceQuota:

```
kubectl get resourcequota pod-demo --namespace=quota-pod-example --output=yaml
```

The output shows that the namespace has a quota of two Pods, and that currently there are no Pods; that is, none of the quota is used.

```
spec:  
  hard:  
    pods: "2"  
status:  
  hard:  
    pods: "2"  
  used:  
    pods: "0"
```

Here is the configuration file for a Deployment:

[admin/resource/quota-pod-deployment.yaml](#)  


```
apiVersion: apps/v1  
kind: Deployment  
metadata:  
  name: pod-quota-demo  
spec:  
  selector:  
    matchLabels:  
      purpose: quota-demo  
  replicas: 3  
  template:  
    metadata:  
      labels:  
        purpose: quota-demo  
    spec:  
      containers:  
      - name: pod-quota-demo  
        image: nginx
```

In the configuration file, `replicas: 3` tells Kubernetes to attempt to create three Pods, all running the same application.

Create the Deployment:

```
kubectl apply -f https://k8s.io/examples/admin/resource/quota-pod-deployment.yaml --namespace=quota-pod-example
```

View detailed information about the Deployment:

```
kubectl get deployment pod-quota-demo --namespace=quota-pod-example --output=yaml
```

The output shows that even though the Deployment specifies three replicas, only two Pods were created because of the quota.

```
spec:  
  replicas: 3  
status:  
  availableReplicas: 2  
lastUpdateTime: 2017-07-07T20:57:05Z  
    message: 'unable to create pods: pods "pod-quota-demo-1650323038-" is forbidden:  
      exceeded quota: pod-demo, requested: pods=1, used: pods=2,  
      limited: pods=2'
```

## Clean up

Delete your namespace:

```
kubectl delete namespace quota-pod-example
```

## What's next

### For cluster administrators

- [Configure Default Memory Requests and Limits for a Namespace](#)
- [Configure Default CPU Requests and Limits for a Namespace](#)
- [Configure Minimum and Maximum Memory Constraints for a Namespace](#)
- [Configure Minimum and Maximum CPU Constraints for a Namespace](#)
- [Configure Memory and CPU Quotas for a Namespace](#)
- [Configure Quotas for API Objects](#)

### For app developers

- [Assign Memory Resources to Containers and Pods](#)

[Assign CPU Resources to Containers and Pods](#)

- [Configure Quality of Service for Pods](#)

## 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](#).

Last modified July 29, 2020 at 9:40 PM PST: [Fix broken links to pages under /en/docs/tasks/administer-cluster/manage-resources/ \(36d9239fb\)](#)  
[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Create a namespace](#)
- [Create a ResourceQuota](#)
- [Clean up](#)
- [What's next](#)
  - [For cluster administrators](#)
  - [For app developers](#)

## Install a Network Policy Provider

---

[Use Calico for NetworkPolicy](#)

[Use Cilium for NetworkPolicy](#)

[Use Kube-router for NetworkPolicy](#)

[Romana for NetworkPolicy](#)

[Weave Net for NetworkPolicy](#)

## Use Calico for NetworkPolicy

This page shows a couple of quick ways to create a Calico cluster on Kubernetes.

# **Before you begin**

Decide whether you want to deploy a [cloud](#) or [local](#) cluster.

## **Creating a Calico cluster with Google Kubernetes Engine (GKE)**

**Prerequisite:** [gcloud](#).

1. To launch a GKE cluster with Calico, just include the `--enable-network-policy` flag.

### **Syntax**

```
gcloud container clusters create [CLUSTER_NAME] --enable-network-policy
```

### **Example**

```
gcloud container clusters create my-calico-cluster --enable-network-policy
```

2. To verify the deployment, use the following command.

```
kubectl get pods --namespace=kube-system
```

The Calico pods begin with `calico`. Check to make sure each one has a status of `Running`.

## **Creating a local Calico cluster with kubeadm**

To get a local single-host Calico cluster in fifteen minutes using kubeadm, refer to the [Calico Quickstart](#).

## **What's next**

Once your cluster is running, you can follow the [Declare Network Policy](#) to try out Kubernetes NetworkPolicy.

## **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](#).

Last modified May 30, 2020 at 3:10 PM PST: [add en pages \(ecc27bbbe\)](#)

- [Before you begin](#)
- [Creating a Calico cluster with Google Kubernetes Engine \(GKE\)](#)
- [Creating a local Calico cluster with kubeadm](#)
- [What's next](#)

# Use Cilium for NetworkPolicy

This page shows how to use Cilium for NetworkPolicy.

For background on Cilium, read the [Introduction to Cilium](#).

## Before you begin

You need to have a Kubernetes cluster, and the `kubectl` command-line tool must be configured to communicate with your cluster. If you do not already have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:

- [Katacoda](#)
- [Play with Kubernetes](#)

To check the version, enter `kubectl version`.

## Deploying Cilium on Minikube for Basic Testing

To get familiar with Cilium easily you can follow the [Cilium Kubernetes Getting Started Guide](#) to perform a basic DaemonSet installation of Cilium in minikube.

To start minikube, minimal version required is  $\geq v1.3.1$ , run the with the following arguments:

```
minikube version
```

```
minikube version: v1.3.1
```

```
minikube start --network-plugin=cni --memory=4096
```

Mount the BPF filesystem:

```
minikube ssh -- sudo mount bpffs -t bpf /sys/fs/bpf
```

For minikube you can deploy this simple "all-in-one" YAML file that includes DaemonSet configurations for Cilium as well as appropriate RBAC settings:

```
kubectl create -f https://raw.githubusercontent.com/cilium/cilium/v1.8/install/kubernetes/quick-install.yaml
```

```
configmap/cilium-config created
serviceaccount/cilium created
serviceaccount/cilium-operator created
clusterrole.rbac.authorization.k8s.io/cilium created
clusterrole.rbac.authorization.k8s.io/cilium-operator created
clusterrolebinding.rbac.authorization.k8s.io/cilium created
clusterrolebinding.rbac.authorization.k8s.io/cilium-operator
created
daemonset.apps/cilium create
deployment.apps/cilium-operator created
```

The remainder of the Getting Started Guide explains how to enforce both L3/L4 (i.e., IP address + port) security policies, as well as L7 (e.g., HTTP) security policies using an example application.

## **Deploying Cilium for Production Use**

For detailed instructions around deploying Cilium for production, see: [Cilium Kubernetes Installation Guide](#) This documentation includes detailed requirements, instructions and example production DaemonSet files.

## **Understanding Cilium components**

Deploying a cluster with Cilium adds Pods to the `kube-system` namespace. To see this list of Pods run:

```
kubectl get pods --namespace=kube-system
```

You'll see a list of Pods similar to this:

NAME	READY	STATUS	RESTARTS	AGE
cilium-6rxbd	1/1	Running	0	1m
...				

A *cilium* Pod runs on each node in your cluster and enforces network policy on the traffic to/from Pods on that node using Linux BPF.

## What's next

Once your cluster is running, you can follow the [Declare Network Policy](#) to try out Kubernetes NetworkPolicy with Cilium. Have fun, and if you have questions, contact us using the [Cilium Slack Channel](#).

## 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](#).

Last modified September 14, 2020 at 10:57 PM PST: [Fix outbound link to Cilium K8s Installation Guide \(2d7b6b85f\)](#)

[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Deploying Cilium on Minikube for Basic Testing](#)
- [Deploying Cilium for Production Use](#)
- [Understanding Cilium components](#)
- [What's next](#)

# Use Kube-router for NetworkPolicy

This page shows how to use [Kube-router](#) for NetworkPolicy.

## Before you begin

You need to have a Kubernetes cluster running. If you do not already have a cluster, you can create one by using any of the cluster installers like Kops, Bootkube, Kubeadm etc.

## Installing Kube-router addon

The Kube-router Addon comes with a Network Policy Controller that watches Kubernetes API server for any NetworkPolicy and pods updated and

configures iptables rules and ipsets to allow or block traffic as directed by the policies. Please follow the [trying Kube-router with cluster installers](#) guide to install Kube-router addon.

## What's next

Once you have installed the Kube-router addon, you can follow the [Declare Network Policy](#) to try out Kubernetes NetworkPolicy.

## 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](#).

Last modified May 30, 2020 at 3:10 PM PST: [add en pages \(ecc27bbbe\)](#)  
[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Installing Kube-router addon](#)
- [What's next](#)

## Romana for NetworkPolicy

This page shows how to use Romana for NetworkPolicy.

### Before you begin

Complete steps 1, 2, and 3 of the [kubeadm getting started guide](#).

### Installing Romana with kubeadm

Follow the [containerized installation guide](#) for kubeadm.

### Applying network policies

To apply network policies use one of the following:

- [Romana network policies](#).
  - [Example of Romana network policy](#).
- [The NetworkPolicy API](#).

## **What's next**

Once you have installed Romana, you can follow the [Declare Network Policy](#) to try out Kubernetes NetworkPolicy.

## **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](#).

Last modified October 22, 2020 at 3:01 PM PST: [Fix links in the tasks section \(740eb340d\)](#)

[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Installing Romana with kubeadm](#)
- [Applying network policies](#)
- [What's next](#)

## **Weave Net for NetworkPolicy**

This page shows how to use Weave Net for NetworkPolicy.

### **Before you begin**

You need to have a Kubernetes cluster. Follow the [kubeadm getting started guide](#) to bootstrap one.

### **Install the Weave Net addon**

Follow the [Integrating Kubernetes via the Addon](#) guide.

The Weave Net addon for Kubernetes comes with a [Network Policy Controller](#) that automatically monitors Kubernetes for any NetworkPolicy annotations on all namespaces and configures iptables rules to allow or block traffic as directed by the policies.

### **Test the installation**

Verify that the weave works.

Enter the following command:

```
kubectl get pods -n kube-system -o wide
```

The output is similar to this:

NAME	RESTARTS	AGE	IP	READY NODE	STATUS
weave-net-1t1qg	0	9d	192.168.2.10	2/2 worknode3	Running
weave-net-231d7	1	7d	10.2.0.17	2/2 worknodegpu	Running
weave-net-7nmwt	3	9d	192.168.2.131	2/2 masternode	Running
weave-net-pmw8w	0	9d	192.168.2.216	2/2 worknode2	Running

Each Node has a weave Pod, and all Pods are Running and 2/2 READY. (2/2 means that each Pod has weave and weave-npc.)

## What's next

Once you have installed the Weave Net addon, you can follow the [Declare Network Policy](#) to try out Kubernetes NetworkPolicy. If you have any question, contact us at [#weave-community on Slack](#) or [Weave User Group](#).

## 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](#).

Last modified October 22, 2020 at 3:01 PM PST: [Fix links in the tasks section \(740eb340d\)](#)

[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Install the Weave Net addon](#)
- [Test the installation](#)
- [What's next](#)

# Access Clusters Using the Kubernetes API

This page shows how to access clusters using the Kubernetes API.

# **Before you begin**

You need to have a Kubernetes cluster, and the `kubectl` command-line tool must be configured to communicate with your cluster. If you do not already have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:

- [Katacoda](#)
- [Play with Kubernetes](#)

To check the version, enter `kubectl version`.

# **Accessing the Kubernetes API**

## **Accessing for the first time with `kubectl`**

When accessing the Kubernetes API for the first time, use the Kubernetes command-line tool, `kubectl`.

To access a cluster, you need to know the location of the cluster and have credentials to access it. Typically, this is automatically set-up when you work through a [Getting started guide](#), or someone else setup the cluster and provided you with credentials and a location.

Check the location and credentials that `kubectl` knows about with this command:

```
kubectl config view
```

Many of the [examples](#) provide an introduction to using `kubectl`. Complete documentation is found in the [kubectl manual](#).

## **Directly accessing the REST API**

`kubectl` handles locating and authenticating to the API server. If you want to directly access the REST API with an http client like `curl` or `wget`, or a browser, there are multiple ways you can locate and authenticate against the API server:

1. Run `kubectl` in proxy mode (recommended). This method is recommended, since it uses the stored apiserver location and verifies the identity of the API server using a self-signed cert. No man-in-the-middle (MITM) attack is possible using this method.

2. Alternatively, you can provide the location and credentials directly to the http client. This works with client code that is confused by proxies. To protect against man in the middle attacks, you'll need to import a root cert into your browser.

Using the Go or Python client libraries provides accessing kubectl in proxy mode.

## Using kubectl proxy

The following command runs kubectl in a mode where it acts as a reverse proxy. It handles locating the API server and authenticating.

Run it like this:

```
kubectl proxy --port=8080 &
```

See [kubectl proxy](#) for more details.

Then you can explore the API with curl, wget, or a browser, like so:

```
curl http://localhost:8080/api/
```

The output is similar to this:

```
{  
  "versions": [  
    "v1"  
,  
  "serverAddressByClientCIDRs": [  
    {  
      "clientCIDR": "0.0.0.0/0",  
      "serverAddress": "10.0.1.149:443"  
    }  
  ]  
}
```

## Without kubectl proxy

It is possible to avoid using kubectl proxy by passing an authentication token directly to the API server, like this:

Using grep/cut approach:

```
# Check all possible clusters, as your .KUBECONFIG may have
multiple contexts:
kubectl config view -o jsonpath='{"Cluster name\tServer\n"}'
{range .clusters[*]}{.name}{"\t"}{.cluster.server}{"\n"}{end}

# Select name of cluster you want to interact with from above
output:
export CLUSTER_NAME="some_server_name"

# Point to the API server referring the cluster name
APISERVER=$(kubectl config view -o jsonpath=".clusters[?
(@.name==\"$CLUSTER_NAME\")].cluster.server")

# Gets the token value
TOKEN=$(kubectl get secrets -o jsonpath=".items[?
(@.metadata.annotations['kubernetes\\.io/service-
account\\.name']=='default')].data.token" | base64 --decode)

# Explore the API with TOKEN
curl -X GET $APISERVER/api --header "Authorization: Bearer
$TOKEN" --insecure
```

The output is similar to this:

```
{
  "kind": "APIVersions",
  "versions": [
    "v1"
  ],
  "serverAddressByClientCIDRs": [
    {
      "clientCIDR": "0.0.0.0/0",
      "serverAddress": "10.0.1.149:443"
    }
  ]
}
```

Using jsonpath approach:

```
APISERVER=$(kubectl config view --minify -o jsonpath='{"clusters[0].cluster.server"}')
TOKEN=$(kubectl get secret $(kubectl get serviceaccount default -o jsonpath='{"secrets[0].name"}') -o jsonpath='{"data.token}"' | base64 --decode )
curl $APISERVER/api --header "Authorization: Bearer $TOKEN" --
insecure
```

```
{
  "kind": "APIVersions",
  "versions": [
    "v1"
  ],
  "serverAddressByClientCIDRs": [
    {
      "clientCIDR": "0.0.0.0/0",
      "serverAddress": "10.0.1.149:443"
    }
  ]
}
```

The above example uses the `--insecure` flag. This leaves it subject to MITM attacks. When kubectl accesses the cluster it uses a stored root certificate and client certificates to access the server. (These are installed in the `~/.kube` directory). Since cluster certificates are typically self-signed, it may take special configuration to get your http client to use root certificate.

On some clusters, the API server does not require authentication; it may serve on localhost, or be protected by a firewall. There is not a standard for this. [Controlling Access to the Kubernetes API](#) describes how you can configure this as a cluster administrator.

## **Programmatic access to the API**

Kubernetes officially supports client libraries for [Go](#), [Python](#), [Java](#), [dotnet](#), [Javascript](#), and [Haskell](#). There are other client libraries that are provided and maintained by their authors, not the Kubernetes team. See [client libraries](#) for accessing the API from other languages and how they authenticate.

### **Go client**

- To get the library, run the following command: `go get k8s.io/client-go@kubernetes-<kubernetes-version-number>` See <https://github.com/kubernetes/client-go/releases> to see which versions are supported.
- Write an application atop of the client-go clients.

**Note:** client-go defines its own API objects, so if needed, import API definitions from client-go rather than from the main repository. For example, `import "k8s.io/client-go/kubernetes"` is correct.

The Go client can use the same [kubeconfig file](#) as the `kubectl` CLI does to locate and authenticate to the API server. See this [example](#):

```
package main

import (
    "context"
    "fmt"
    "k8s.io/apimachinery/pkg/apis/meta/v1"
    "k8s.io/client-go/kubernetes"
    "k8s.io/client-go/tools/clientcmd"
)

func main() {
    // uses the current context in kubeconfig
    // path-to-kubeconfig -- for example, /root/.kube/config
    config, _ := clientcmd.BuildConfigFromFlags("", "<path-to-
kubeconfig>")
    // creates the clientset
    clientset, _ := kubernetes.NewForConfig(config)
    // access the API to list pods
    pods, _ := clientset.CoreV1().Pods("").List(context.TODO(),
v1.ListOptions{})
    fmt.Printf("There are %d pods in the cluster\n", len(pods.Items
))
}
```

If the application is deployed as a Pod in the cluster, see [Accessing the API from within a Pod](#).

## Python client

To use [Python client](#), run the following command: `pip install kubernetes`. See [Python Client Library page](#) for more installation options.

The Python client can use the same [kubeconfig file](#) as the `kubectl` CLI does to locate and authenticate to the API server. See this [example](#):

```
from kubernetes import client, config

config.load_kube_config()

v1=client.CoreV1Api()
print("Listing pods with their IPs:")
ret = v1.list_pod_for_all_namespaces(watch=False)
for i in ret.items:
```

```
    print("%s\t%s\t%s" % (i.status.pod_ip, i.metadata.namespace,
i.metadata.name))
```

## Java client

- To install the [Java Client](#), simply execute :

```
# Clone java library
git clone --recursive https://github.com/kubernetes-client/java

# Installing project artifacts, POM etc:
cd java
mvn install
```

See <https://github.com/kubernetes-client/java/releases> to see which versions are supported.

The Java client can use the same [kubeconfig file](#) as the kubectl CLI does to locate and authenticate to the API server. See this [example](#):

```
package io.kubernetes.client.examples;

import io.kubernetes.client.ApiClient;
import io.kubernetes.client.ApiException;
import io.kubernetes.client.Configuration;
import io.kubernetes.client.apis.CoreV1Api;
import io.kubernetes.client.models.V1Pod;
import io.kubernetes.client.models.V1PodList;
import io.kubernetes.client.util.ClientBuilder;
import io.kubernetes.client.util.KubeConfig;
import java.io.FileReader;
import java.io.IOException;

/**
 * A simple example of how to use the Java API from an
application outside a kubernetes cluster
 *
 * <p>Easiest way to run this: mvn exec:java
 *
Dexec.mainClass="io.kubernetes.client.examples.KubeConfigFileClientExample"
*
*/
public class KubeConfigFileClientExample {
    public static void main(String[] args) throws IOException,
    ApiException {
```

```

// file path to your KubeConfig
String kubeConfigPath = "~/.kube/config";

// loading the out-of-cluster config, a kubeconfig from file-
system
ApiClient client =
    ClientBuilder.kubeconfig(KubeConfig.loadKubeConfig(new
FileReader(kubeConfigPath))).build();

// set the global default api-client to the in-cluster one
from above
Configuration.setDefaultApiClient(client);

// the CoreV1Api loads default api-client from global
configuration.
CoreV1Api api = new CoreV1Api();

// invokes the CoreV1Api client
V1PodList list = api.listPodForAllNamespaces(null, null,
null, null, null, null, null, null);
System.out.println("Listing all pods: ");
for (V1Pod item : list.getItems()) {
    System.out.println(item.getMetadata().getName());
}
}
}
}

```

## dotnet client

To use [dotnet client](#), run the following command: `dotnet add package KubernetesClient --version 1.6.1` See [dotnet Client Library page](#) for more installation options. See <https://github.com/kubernetes-client/csharp/releases> to see which versions are supported.

The dotnet client can use the same [kubeconfig file](#) as the kubectl CLI does to locate and authenticate to the API server. See this [example](#):

```

using System;
using k8s;

namespace simple
{
    internal class PodList
    {
        private static void Main(string[] args)
        {
            var config =
KubernetesClientConfiguration.BuildDefaultConfig();

```

```

IKubernetes client = new Kubernetes(config);
Console.WriteLine("Starting Request!");

var list = client.ListNamespacedPod("default");
foreach (var item in list.Items)
{
    Console.WriteLine(item.Metadata.Name);
}
if (list.Items.Count == 0)
{
    Console.WriteLine("Empty!");
}
}
}
}

```

## **JavaScript client**

To install [JavaScript client](#), run the following command: `npm install @kubernetes/client-node`. See <https://github.com/kubernetes-client/javascript/releases> to see which versions are supported.

The JavaScript client can use the same [kubeconfig file](#) as the `kubectl` CLI does to locate and authenticate to the API server. See this [example](#):

```

const k8s = require('@kubernetes/client-node');

const kc = new k8s.KubeConfig();
kc.loadFromDefault();

const k8sApi = kc.makeApiClient(k8s.CoreV1Api);

k8sApi.listNamespacedPod('default').then((res) => {
    console.log(res.body);
});

```

## **Haskell client**

See <https://github.com/kubernetes-client/haskell/releases> to see which versions are supported.

The [Haskell client](#) can use the same [kubeconfig file](#) as the `kubectl` CLI does to locate and authenticate to the API server. See this [example](#):

```

exampleWithKubeConfig :: IO ()
exampleWithKubeConfig = do
    oidcCache <- atomically $ newTVar $ Map.fromList []
    (mgr, kcfg) <- mkKubeClientConfig oidcCache $ KubeConfigFile
    "/path/to/kubeconfig"
    dispatchMime
        mgr
        kcfg
        (CoreV1.listPodForAllNamespaces (Accept MimeJSON))
    >>= print

```

## **Accessing the API from within a Pod**

*When accessing the API from within a Pod, locating and authenticating to the API server are slightly different to the external client case described above.*

*The easiest way to use the Kubernetes API from a Pod is to use one of the official [client libraries](#). These libraries can automatically discover the API server and authenticate.*

### **Using Official Client Libraries**

*From within a Pod, the recommended ways to connect to the Kubernetes API are:*

- *For a Go client, use the official [Go client library](#). The `rest.InClusterConfig()` function handles API host discovery and authentication automatically. See [an example here](#).*
- *For a Python client, use the official [Python client library](#). The `config.load_incluster_config()` function handles API host discovery and authentication automatically. See [an example here](#).*
- *There are a number of other libraries available, please refer to the [Client Libraries](#) page.*

*In each case, the service account credentials of the Pod are used to communicate securely with the API server.*

### **Directly accessing the REST API**

*While running in a Pod, the Kubernetes apiserver is accessible via a Service named `kubernetes` in the default namespace. Therefore, Pods can use the*

`kubernetes.default.svc` hostname to query the API server. Official client libraries do this automatically.

The recommended way to authenticate to the API server is with a [service account](#) credential. By default, a Pod is associated with a service account, and a credential (token) for that service account is placed into the filesystem tree of each container in that Pod, at `/var/run/secrets/kubernetes.io/serviceaccount/token`.

If available, a certificate bundle is placed into the filesystem tree of each container at `/var/run/secrets/kubernetes.io/serviceaccount/ca.crt`, and should be used to verify the serving certificate of the API server.

Finally, the default namespace to be used for namespaced API operations is placed in a file at `/var/run/secrets/kubernetes.io/serviceaccount/namespace` in each container.

## Using `kubectl proxy`

If you would like to query the API without an official client library, you can run `kubectl proxy` as the [command](#) of a new sidecar container in the Pod. This way, `kubectl proxy` will authenticate to the API and expose it on the `localhost` interface of the Pod, so that other containers in the Pod can use it directly.

## Without using a proxy

It is possible to avoid using the `kubectl proxy` by passing the authentication token directly to the API server. The internal certificate secures the connection.

```
# Point to the internal API server hostname
APISERVER=https://kubernetes.default.svc

# Path to ServiceAccount token
SERVICEACCOUNT=/var/run/secrets/kubernetes.io/serviceaccount

# Read this Pod's namespace
NAMESPACE=$(cat ${SERVICEACCOUNT}/namespace)

# Read the ServiceAccount bearer token
TOKEN=$(cat ${SERVICEACCOUNT}/token)

# Reference the internal certificate authority (CA)
CACERT=${SERVICEACCOUNT}/ca.crt
```

```
# Explore the API with TOKEN
curl --cacert ${CACERT} --header "Authorization: Bearer ${TOKEN}"
-X GET ${APISERVER}/api
```

The output will be similar to this:

```
{
  "kind": "APIVersions",
  "versions": [
    "v1"
  ],
  "serverAddressByClientCIDRs": [
    {
      "clientCIDR": "0.0.0.0/0",
      "serverAddress": "10.0.1.149:443"
    }
  ]
}
```

## 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](#).

Last modified October 13, 2020 at 12:41 AM PST: [Transfer "Controlling Access to the Kubernetes API" to the Concepts section \(78351ecaf\)](#)  
[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Accessing the Kubernetes API](#)
  - [Accessing for the first time with kubectl](#)
  - [Directly accessing the REST API](#)
  - [Programmatic access to the API](#)
  - [Accessing the API from within a Pod](#)

# Access Services Running on Clusters

This page shows how to connect to services running on the Kubernetes cluster.

# **Before you begin**

You need to have a Kubernetes cluster, and the `kubectl` command-line tool must be configured to communicate with your cluster. If you do not already have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:

- [Katacoda](#)
- [Play with Kubernetes](#)

To check the version, enter `kubectl version`.

# **Accessing services running on the cluster**

In Kubernetes, [nodes](#), [pods](#) and [services](#) all have their own IPs. In many cases, the node IPs, pod IPs, and some service IPs on a cluster will not be routable, so they will not be reachable from a machine outside the cluster, such as your desktop machine.

## **Ways to connect**

You have several options for connecting to nodes, pods and services from outside the cluster:

- Access services through public IPs.
  - Use a service with type `NodePort` or `LoadBalancer` to make the service reachable outside the cluster. See the [services](#) and [kubectl expose](#) documentation.
  - Depending on your cluster environment, this may just expose the service to your corporate network, or it may expose it to the internet. Think about whether the service being exposed is secure. Does it do its own authentication?
  - Place pods behind services. To access one specific pod from a set of replicas, such as for debugging, place a unique label on the pod and create a new service which selects this label.
  - In most cases, it should not be necessary for application developer to directly access nodes via their nodeIPs.
- Access services, nodes, or pods using the Proxy Verb.
  - Does apiserver authentication and authorization prior to accessing the remote service. Use this if the services are not secure enough to expose to the internet, or to gain access to ports on the node IP, or for debugging.
  - Proxies may cause problems for some web applications.
  - Only works for HTTP/HTTPS.
  - Described [here](#).

- Access from a node or pod in the cluster.
  - Run a pod, and then connect to a shell in it using [kubectl exec](#). Connect to other nodes, pods, and services from that shell.
  - Some clusters may allow you to ssh to a node in the cluster. From there you may be able to access cluster services. This is a non-standard method, and will work on some clusters but not others. Browsers and other tools may or may not be installed. Cluster DNS may not work.

## ***Discovering builtin services***

*Typically, there are several services which are started on a cluster by kube-system. Get a list of these with the kubectl cluster-info command:*

```
kubectl cluster-info
```

*The output is similar to this:*

```
Kubernetes master is running at https://104.197.5.247
elasticsearch-logging is running at https://104.197.5.247/api/v1/
namespaces/kube-system/services/elasticsearch-logging/proxy
kibana-logging is running at https://104.197.5.247/api/v1/
namespaces/kube-system/services/kibana-logging/proxy
kube-dns is running at https://104.197.5.247/api/v1/namespaces/
kube-system/services/kube-dns/proxy
grafana is running at https://104.197.5.247/api/v1/namespaces/
kube-system/services/monitoring-grafana/proxy
heapster is running at https://104.197.5.247/api/v1/namespaces/
kube-system/services/monitoring-heapster/proxy
```

*This shows the proxy-verb URL for accessing each service. For example, this cluster has cluster-level logging enabled (using Elasticsearch), which can be reached at https://104.197.5.247/api/v1/namespaces/kube-system/
services/elasticsearch-logging/proxy/ if suitable credentials are passed, or through a kubectl proxy at, for example: http://localhost:8080/api/v1/namespaces/kube-system/services/elasticsearch-logging/proxy/.*

**Note:** See [Access Clusters Using the Kubernetes API](#) for how to pass credentials or use kubectl proxy.

## ***Manually constructing apiserver proxy URLs***

As mentioned above, you use the `kubectl cluster-info` command to retrieve the service's proxy URL. To create proxy URLs that include service endpoints, suffixes, and parameters, you simply append to the service's proxy URL: `http://kubernetes_master_address/api/v1/namespaces/name_space_name/services/[https:]service_name[:port_name]/proxy`

If you haven't specified a name for your port, you don't have to specify `port_name` in the URL.

### Examples

- To access the Elasticsearch service endpoint `_search?q=user:kimchy`, you would use:

```
http://104.197.5.247/api/v1/namespaces/kube-system/services/elasticsearch-logging/proxy/_search?q=user:kimchy
```

- To access the Elasticsearch cluster health information `_cluster/health?pretty=true`, you would use:

```
https://104.197.5.247/api/v1/namespaces/kube-system/services/elasticsearch-logging/proxy/_cluster/health?pretty=true
```

The health information is similar to this:

```
{  
    "cluster_name" : "kubernetes_logging",  
    "status" : "yellow",  
    "timed_out" : false,  
    "number_of_nodes" : 1,  
    "number_of_data_nodes" : 1,  
    "active_primary_shards" : 5,  
    "active_shards" : 5,  
    "relocating_shards" : 0,  
    "initializing_shards" : 0,  
    "unassigned_shards" : 5  
}
```

- To access the https Elasticsearch service health information `_cluster/health?pretty=true`, you would use:

```
https://104.197.5.247/api/v1/namespaces/kube-system/services/https:elasticsearch-logging/proxy/_cluster/health?pretty=true
```

### Using web browsers to access services running on the cluster

You may be able to put an apiserver proxy URL into the address bar of a browser. However:

- Web browsers cannot usually pass tokens, so you may need to use basic (password) auth. Apiserver can be configured to accept basic auth, but your cluster may not be configured to accept basic auth.
- Some web apps may not work, particularly those with client side javascript that construct URLs in a way that is unaware of the proxy path prefix.

## 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](#).

Last modified August 07, 2020 at 8:40 PM PST: [Tune links in tasks section \(2/2\) \(92ae1a9cf\)](#)

[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Accessing services running on the cluster](#)
  - [Ways to connect](#)
  - [Discovering builtin services](#)

# Advertise Extended Resources for a Node

This page shows how to specify extended resources for a Node. Extended resources allow cluster administrators to advertise node-level resources that would otherwise be unknown to Kubernetes.

## Before you begin

You need to have a Kubernetes cluster, and the `kubectl` command-line tool must be configured to communicate with your cluster. If you do not already have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:

- [Katacoda](#)
- [Play with Kubernetes](#)

To check the version, enter `kubectl version`.

## **Get the names of your Nodes**

```
kubectl get nodes
```

Choose one of your Nodes to use for this exercise.

## **Advertise a new extended resource on one of your Nodes**

To advertise a new extended resource on a Node, send an HTTP PATCH request to the Kubernetes API server. For example, suppose one of your Nodes has four dongles attached. Here's an example of a PATCH request that advertises four dongle resources for your Node.

```
PATCH /api/v1/nodes/<your-node-name>/status HTTP/1.1  
Accept: application/json  
Content-Type: application/json-patch+json  
Host: k8s-master:8080
```

```
[  
  {  
    "op": "add",  
    "path": "/status/capacity/example.com~1dongle",  
    "value": "4"  
  }  
]
```

Note that Kubernetes does not need to know what a dongle is or what a dongle is for. The preceding PATCH request just tells Kubernetes that your Node has four things that you call dongles.

Start a proxy, so that you can easily send requests to the Kubernetes API server:

```
kubectl proxy
```

In another command window, send the HTTP PATCH request. Replace <your-node-name> with the name of your Node:

```
curl --header "Content-Type: application/json-patch+json" \  
--request PATCH \  
--data '[{"op": "add", "path": "/status/capacity/
```

```
example.com~1dongle", "value": "4"}]' \
http://localhost:8001/api/v1/nodes/<your-node-name>/status
```

**Note:** In the preceding request, `~1` is the encoding for the character `/` in the patch path. The operation path value in JSON-Patch is interpreted as a JSON-Pointer. For more details, see [IETF RFC 6901](#), section 3.

The output shows that the Node has a capacity of 4 dongles:

```
"capacity": {
  "cpu": "2",
  "memory": "2049008Ki",
  "example.com/dongle": "4",
```

Describe your Node:

```
kubectl describe node <your-node-name>
```

Once again, the output shows the dongle resource:

```
Capacity:
cpu: 2
memory: 2049008Ki
example.com/dongle: 4
```

Now, application developers can create Pods that request a certain number of dongles. See [Assign Extended Resources to a Container](#).

## Discussion

Extended resources are similar to memory and CPU resources. For example, just as a Node has a certain amount of memory and CPU to be shared by all components running on the Node, it can have a certain number of dongles to be shared by all components running on the Node. And just as application developers can create Pods that request a certain amount of memory and CPU, they can create Pods that request a certain number of dongles.

Extended resources are opaque to Kubernetes; Kubernetes does not know anything about what they are. Kubernetes knows only that a Node has a certain number of them. Extended resources must be advertised in integer amounts. For example, a Node can advertise four dongles, but not 4.5 dongles.

## **Storage example**

Suppose a Node has 800 GiB of a special kind of disk storage. You could create a name for the special storage, say example.com/special-storage. Then you could advertise it in chunks of a certain size, say 100 GiB. In that case, your Node would advertise that it has eight resources of type example.com/special-storage.

**Capacity:**

```
example.com/special-storage: 8
```

If you want to allow arbitrary requests for special storage, you could advertise special storage in chunks of size 1 byte. In that case, you would advertise 800Gi resources of type example.com/special-storage.

**Capacity:**

```
example.com/special-storage: 800Gi
```

Then a Container could request any number of bytes of special storage, up to 800Gi.

## **Clean up**

Here is a PATCH request that removes the dongle advertisement from a Node.

```
PATCH /api/v1/nodes/<your-node-name>/status HTTP/1.1
Accept: application/json
Content-Type: application/json-patch+json
Host: k8s-master:8080

[{"op": "remove", "path": "/status/capacity/example.com~1dongle"},]
```

Start a proxy, so that you can easily send requests to the Kubernetes API server:

```
kubectl proxy
```

In another command window, send the HTTP PATCH request. Replace <your-node-name> with the name of your Node:

```
curl --header "Content-Type: application/json-patch+json" \
--request PATCH \
--data '[{"op": "remove", "path": "/status/capacity/
example.com~1dongle"}]' \
http://localhost:8001/api/v1/nodes/<your-node-name>/status
```

Verify that the dongle advertisement has been removed:

```
kubectl describe node <your-node-name> | grep dongle
```

(you should not see any output)

## What's next

### For application developers

- [Assign Extended Resources to a Container](#)

### For cluster administrators

- [Configure Minimum and Maximum Memory Constraints for a Namespace](#)
- [Configure Minimum and Maximum CPU Constraints for a 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](#).

Last modified July 29, 2020 at 9:40 PM PST: [Fix broken links to pages under /en/docs/tasks/administer-cluster/manage-resources/ \(36d9239fb\)](#)

- [Before you begin](#)
- [Get the names of your Nodes](#)
- [Advertise a new extended resource on one of your Nodes](#)
- [Discussion](#)
  - [Storage example](#)
- [Clean up](#)
- [What's next](#)
  - [For application developers](#)
  - [For cluster administrators](#)

# Autoscale the DNS Service in a Cluster

This page shows how to enable and configure autoscaling of the DNS service in your Kubernetes cluster.

## Before you begin

- You need to have a Kubernetes cluster, and the `kubectl` command-line tool must be configured to communicate with your cluster. If you do not already have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:

- [Katacoda](#)
- [Play with Kubernetes](#)

To check the version, enter `kubectl version`.

- This guide assumes your nodes use the AMD64 or Intel 64 CPU architecture.
- Make sure [Kubernetes DNS](#) is enabled.

## Determine whether DNS horizontal autoscaling is already enabled

List the [Deployments](#) in your cluster in the `kube-system` [namespace](#):

```
kubectl get deployment --namespace=kube-system
```

The output is similar to this:

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
...				
<code>dns-autoscaler</code>	1/1	1	1	...
...				

If you see "dns-autoscaler" in the output, DNS horizontal autoscaling is already enabled, and you can skip to [Tuning autoscaling parameters](#).

## Get the name of your DNS Deployment

List the DNS deployments in your cluster in the kube-system namespace:

```
kubectl get deployment -l k8s-app=kube-dns --namespace=kube-system
```

The output is similar to this:

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
coredns	2/2	2	2	...
...				

If you don't see a Deployment for DNS services, you can also look for it by name:

```
kubectl get deployment --namespace=kube-system
```

and look for a deployment named `coredns` or `kube-dns`.

Your scale target is

```
Deployment/<your-deployment-name>
```

where `<your-deployment-name>` is the name of your DNS Deployment. For example, if the name of your Deployment for DNS is `coredns`, your scale target is `Deployment/coredns`.

**Note:** CoreDNS is the default DNS service for Kubernetes.

CoreDNS sets the label `k8s-app=kube-dns` so that it can work in clusters that originally used `kube-dns`.

## Enable DNS horizontal autoscaling

In this section, you create a new Deployment. The Pods in the Deployment run a container based on the `cluster-proportional-autoscaler-amd64` image.

Create a file named `dns-horizontal-autoscaler.yaml` with this content:

[admin/dns/dns-horizontal-autoscaler.yaml](#)



```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: dns-autoscaler
  namespace: kube-system
```

```

labels:
  k8s-app: dns-autoscaler
spec:
  selector:
    matchLabels:
      k8s-app: dns-autoscaler
template:
  metadata:
    labels:
      k8s-app: dns-autoscaler
  spec:
    containers:
      - name: autoscaler
        image: k8s.gcr.io/cluster-proportional-autoscaler-amd64:1
.6.0
        resources:
          requests:
            cpu: 20m
            memory: 10Mi
        command:
          - /cluster-proportional-autoscaler
          - --namespace=kube-system
          - --configmap=dns-autoscaler
          - --target=<SCALE_TARGET>
# When cluster is using large nodes (with more cores),
"coresPerReplica" should dominate.
# If using small nodes, "nodesPerReplica" should
dominate.
          - --default-params={"linear":{"coresPerReplica":256,"node
sPerReplica":16,"min":1}}
          - --logtostderr=true
          - --v=2

```

In the file, replace <SCALE\_TARGET> with your scale target.

Go to the directory that contains your configuration file, and enter this command to create the Deployment:

```
kubectl apply -f dns-horizontal-autoscaler.yaml
```

The output of a successful command is:

```
deployment.apps/dns-autoscaler created
```

DNS horizontal autoscaling is now enabled.

## Tune DNS autoscaling parameters

Verify that the dns-autoscaler [ConfigMap](#) exists:

```
kubectl get configmap --namespace=kube-system
```

The output is similar to this:

NAME	DATA	AGE
...		
<i>dns-autoscaler</i>	1	...
...		

Modify the data in the ConfigMap:

```
kubectl edit configmap dns-autoscaler --namespace=kube-system
```

Look for this line:

```
linear: '{"coresPerReplica":256,"min":1,"nodesPerReplica":16}'
```

Modify the fields according to your needs. The "min" field indicates the minimal number of DNS backends. The actual number of backends is calculated using this equation:

```
replicas = max( ceil( cores - 1/coresPerReplica ) , ceil( nodes - 1/nodesPerReplica ) )
```

Note that the values of both `coresPerReplica` and `nodesPerReplica` are floats.

The idea is that when a cluster is using nodes that have many cores, `coresPerReplica` dominates. When a cluster is using nodes that have fewer cores, `nodesPerReplica` dominates.

There are other supported scaling patterns. For details, see [cluster-proportional-autoscaler](#).

## Disable DNS horizontal autoscaling

There are a few options for tuning DNS horizontal autoscaling. Which option to use depends on different conditions.

### Option 1: Scale down the `dns-autoscaler` deployment to 0 replicas

This option works for all situations. Enter this command:

```
kubectl scale deployment --replicas=0 dns-autoscaler --namespace=kube-system
```

The output is:

```
deployment.apps/dns-autoscaler scaled
```

Verify that the replica count is zero:

```
kubectl get rs --namespace=kube-system
```

The output displays 0 in the DESIRED and CURRENT columns:

NAME	DESIRED	CURRENT	READY
AGE			
...			
<code>dns-autoscaler-6b59789fc8</code>	0	0	
0			
...			

## **Option 2: Delete the dns-autoscaler deployment**

This option works if dns-autoscaler is under your own control, which means no one will re-create it:

```
kubectl delete deployment dns-autoscaler --namespace=kube-system
```

The output is:

```
deployment.apps "dns-autoscaler" deleted
```

## **Option 3: Delete the dns-autoscaler manifest file from the master node**

This option works if dns-autoscaler is under control of the (deprecated) [Addon Manager](#), and you have write access to the master node.

Sign in to the master node and delete the corresponding manifest file. The common path for this dns-autoscaler is:

```
/etc/kubernetes/addons/dns-horizontal-autoscaler/dns-horizontal-autoscaler.yaml
```

After the manifest file is deleted, the Addon Manager will delete the dns-autoscaler Deployment.

## **Understanding how DNS horizontal autoscaling works**

- The cluster-proportional-autoscaler application is deployed separately from the DNS service.
- An autoscaler Pod runs a client that polls the Kubernetes API server for the number of nodes and cores in the cluster.
- A desired replica count is calculated and applied to the DNS backends based on the current schedulable nodes and cores and the given scaling parameters.
- The scaling parameters and data points are provided via a ConfigMap to the autoscaler, and it refreshes its parameters table every poll interval to be up to date with the latest desired scaling parameters.

- Changes to the scaling parameters are allowed without rebuilding or restarting the autoscaler Pod.
- The autoscaler provides a controller interface to support two control patterns: linear and ladder.

## What's next

- Read about [Guaranteed Scheduling For Critical Add-On Pods](#).
- Learn more about the [implementation of cluster-proportional-autoscaler](#).

## 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](#).

Last modified October 17, 2020 at 3:21 PM PST: [update kubernetes-incubator references \(a8b6551c2\)](#)

[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Determine whether DNS horizontal autoscaling is already enabled](#)
- [Get the name of your DNS Deployment](#)
- [Enable DNS horizontal autoscaling](#)
- [Tune DNS autoscaling parameters](#)
- [Disable DNS horizontal autoscaling](#)
  - [Option 1: Scale down the dns-autoscaler deployment to 0 replicas](#)
  - [Option 2: Delete the dns-autoscaler deployment](#)
  - [Option 3: Delete the dns-autoscaler manifest file from the master node](#)
- [Understanding how DNS horizontal autoscaling works](#)
- [What's next](#)

# Change the default StorageClass

This page shows how to change the default Storage Class that is used to provision volumes for PersistentVolumeClaims that have no special requirements.

## Before you begin

You need to have a Kubernetes cluster, and the `kubectl` command-line tool must be configured to communicate with your cluster. If you do not already

have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:

- [Katacoda](#)
- [Play with Kubernetes](#)

To check the version, enter `kubectl version`.

## Why change the default storage class?

Depending on the installation method, your Kubernetes cluster may be deployed with an existing `StorageClass` that is marked as default. This default `StorageClass` is then used to dynamically provision storage for `PersistentVolumeClaims` that do not require any specific storage class. See [PersistentVolumeClaim documentation](#) for details.

The pre-installed default `StorageClass` may not fit well with your expected workload; for example, it might provision storage that is too expensive. If this is the case, you can either change the default `StorageClass` or disable it completely to avoid dynamic provisioning of storage.

Simply deleting the default `StorageClass` may not work, as it may be re-created automatically by the addon manager running in your cluster. Please consult the docs for your installation for details about addon manager and how to disable individual addons.

## Changing the default StorageClass

1. List the `StorageClasses` in your cluster:

```
kubectl get storageclass
```

The output is similar to this:

NAME	PROVISIONER	AGE
standard (default)	<code>kubernetes.io/gce-pd</code>	1d
gold	<code>kubernetes.io/gce-pd</code>	1d

The default `StorageClass` is marked by `(default)`.

2. Mark the default `StorageClass` as non-default:

The default `StorageClass` has an annotation `storageclass.kubernetes.io/is-default-class` set to `true`. Any other value or absence of the annotation is interpreted as `false`.

To mark a `StorageClass` as non-default, you need to change its value to `false`:

```
kubectl patch storageclass standard -p '{"metadata": {"annotations":{"storageclass.kubernetes.io/is-default-class":"false"}}}'
```

where `standard` is the name of your chosen `StorageClass`.

### 3. Mark a `StorageClass` as default:

Similarly to the previous step, you need to add/set the annotation `storageclass.kubernetes.io/is-default-class=true`.

```
kubectl patch storageclass gold -p '{"metadata": {"annotations":{"storageclass.kubernetes.io/is-default-class":"true"}}}'
```

Please note that at most one `StorageClass` can be marked as default. If two or more of them are marked as default, a `PersistentVolumeClaim` without `storageClassName` explicitly specified cannot be created.

### 4. Verify that your chosen `StorageClass` is default:

```
kubectl get storageclass
```

The output is similar to this:

NAME	PROVISIONER	AGE
standard	<code>kubernetes.io/gce-pd</code>	<code>1d</code>
gold (default)	<code>kubernetes.io/gce-pd</code>	<code>1d</code>

## What's next

- Learn more about [PersistentVolumes](#).

## 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](#).

Last modified June 27, 2020 at 8:58 PM PST: [Fix issue with links to wrong section name \(427b4ec42\)](#)

[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Why change the default storage class?](#)
- [Changing the default StorageClass](#)

- [What's next](#)

# Change the Reclaim Policy of a PersistentVolume

This page shows how to change the reclaim policy of a Kubernetes PersistentVolume.

## Before you begin

You need to have a Kubernetes cluster, and the `kubectl` command-line tool must be configured to communicate with your cluster. If you do not already have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:

- [Katacoda](#)
- [Play with Kubernetes](#)

To check the version, enter `kubectl version`.

## Why change reclaim policy of a PersistentVolume

PersistentVolumes can have various reclaim policies, including "Retain", "Recycle", and "Delete". For dynamically provisioned PersistentVolumes, the default reclaim policy is "Delete". This means that a dynamically provisioned volume is automatically deleted when a user deletes the corresponding PersistentVolumeClaim. This automatic behavior might be inappropriate if the volume contains precious data. In that case, it is more appropriate to use the "Retain" policy. With the "Retain" policy, if a user deletes a PersistentVolumeClaim, the corresponding PersistentVolume is not be deleted. Instead, it is moved to the Released phase, where all of its data can be manually recovered.

## Changing the reclaim policy of a PersistentVolume

1. List the PersistentVolumes in your cluster:

```
kubectl get pv
```

The output is similar to this:

NAME			CAPACITY
ACCESSMODES	RECLAIMPOLICY	STATUS	CLAIM
STORAGECLASS	REASON	AGE	
pvc-b6efd8da-b7b5-11e6-9d58-0ed433a7dd94		4Gi	
RWO	Delete	Bound	default/claim1
manual		10s	
pvc-b95650f8-b7b5-11e6-9d58-0ed433a7dd94		4Gi	
RWO	Delete	Bound	default/claim2
manual		6s	
pvc-bb3ca71d-b7b5-11e6-9d58-0ed433a7dd94		4Gi	
RWO	Delete	Bound	default/claim3
manual		3s	

This list also includes the name of the claims that are bound to each volume for easier identification of dynamically provisioned volumes.

- Choose one of your PersistentVolumes and change its reclaim policy:

```
kubectl patch pv <your-pv-name> -p '{"spec": {"persistentVolumeReclaimPolicy": "Retain"}}'
```

where `<your-pv-name>` is the name of your chosen PersistentVolume.

**Note:**

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:

```
kubectl patch pv <your-pv-name> -p "{\"spec\":{\"persistentVolumeReclaimPolicy\": \"Retain\"}}"
```

- Verify that your chosen PersistentVolume has the right policy:

```
kubectl get pv
```

The output is similar to this:

NAME			CAPACITY
ACCESSMODES	RECLAIMPOLICY	STATUS	CLAIM
STORAGECLASS	REASON	AGE	
pvc-b6efd8da-b7b5-11e6-9d58-0ed433a7dd94		4Gi	
RWO	Delete	Bound	default/claim1
manual		40s	
pvc-b95650f8-b7b5-11e6-9d58-0ed433a7dd94		4Gi	
RWO	Delete	Bound	default/claim2
manual		36s	
pvc-bb3ca71d-b7b5-11e6-9d58-0ed433a7dd94		4Gi	
RWO	Retain	Bound	default/claim3
manual		33s	

*In the preceding output, you can see that the volume bound to claim default/claim3 has reclaim policy Retain. It will not be automatically deleted when a user deletes claim default/claim3.*

## What's next

- Learn more about [PersistentVolumes](#).
- Learn more about [PersistentVolumeClaims](#).

## Reference

- [PersistentVolume](#)
- [PersistentVolumeClaim](#)
- See the `persistentVolumeReclaimPolicy` field of [PersistentVolumeSpec](#).

## 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](#).

Last modified July 23, 2020 at 12:38 AM PST: [remove backticks from "Change the Reclaim Policy of a PersistentVolume" page \(531a496c1\)](#)  
[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Why change reclaim policy of a PersistentVolume](#)
- [Changing the reclaim policy of a PersistentVolume](#)
- [What's next](#)
  - [Reference](#)

# Cloud Controller Manager Administration

**FEATURE STATE:** Kubernetes v1.11 [beta]

Since cloud providers develop and release at a different pace compared to the Kubernetes project, abstracting the provider-specific code to the [cloud-](#)

[`controller-manager`](#) binary allows cloud vendors to evolve independently from the core Kubernetes code.

The `cloud-controller-manager` can be linked to any cloud provider that satisfies [`cloudprovider.Interface`](#). For backwards compatibility, the [`cloud-controller-manager`](#) provided in the core Kubernetes project uses the same cloud libraries as `kube-controller-manager`. Cloud providers already supported in Kubernetes core are expected to use the in-tree `cloud-controller-manager` to transition out of Kubernetes core.

## Administration

### Requirements

Every cloud has their own set of requirements for running their own cloud provider integration, it should not be too different from the requirements when running `kube-controller-manager`. As a general rule of thumb you'll need:

- *cloud authentication/authorization: your cloud may require a token or IAM rules to allow access to their APIs*
- *kubernetes authentication/authorization: cloud-controller-manager may need RBAC rules set to speak to the kubernetes apiserver*
- *high availability: like kube-controller-manager, you may want a high available setup for cloud controller manager using leader election (on by default).*

### Running `cloud-controller-manager`

Successfully running `cloud-controller-manager` requires some changes to your cluster configuration.

- *`kube-apiserver` and `kube-controller-manager` MUST NOT specify the `--cloud-provider` flag. This ensures that it does not run any cloud specific loops that would be run by cloud controller manager. In the future, this flag will be deprecated and removed.*
- *`kubelet` must run with `--cloud-provider=external`. This is to ensure that the `kubelet` is aware that it must be initialized by the cloud controller manager before it is scheduled any work.*

Keep in mind that setting up your cluster to use cloud controller manager will change your cluster behaviour in a few ways:

- *`kubelets` specifying `--cloud-provider=external` will add a taint node. `cloudprovider.kubernetes.io/uninitialized` with an effect `NoSchedule` during initialization. This marks the node as needing a second initialization from an external controller before it can be scheduled work. Note that in the event that cloud controller manager is not available, new nodes in the cluster will be left unschedulable. The taint is important since the scheduler may require cloud specific information*

*about nodes such as their region or type (high cpu, gpu, high memory, spot instance, etc).*

- *cloud information about nodes in the cluster will no longer be retrieved using local metadata, but instead all API calls to retrieve node information will go through cloud controller manager. This may mean you can restrict access to your cloud API on the kubelets for better security. For larger clusters you may want to consider if cloud controller manager will hit rate limits since it is now responsible for almost all API calls to your cloud from within the cluster.*

*The cloud controller manager can implement:*

- *Node controller - responsible for updating kubernetes nodes using cloud APIs and deleting kubernetes nodes that were deleted on your cloud.*
- *Service controller - responsible for loadbalancers on your cloud against services of type LoadBalancer.*
- *Route controller - responsible for setting up network routes on your cloud*
- *any other features you would like to implement if you are running an out-of-tree provider.*

## Examples

*If you are using a cloud that is currently supported in Kubernetes core and would like to adopt cloud controller manager, see the [cloud controller manager in kubernetes core](#).*

*For cloud controller managers not in Kubernetes core, you can find the respective projects in repositories maintained by cloud vendors or by SIGs.*

*For providers already in Kubernetes core, you can run the in-tree cloud controller manager as a DaemonSet in your cluster, use the following as a guideline:*

[admin/cloud/ccm-example.yaml](#)



```
# This is an example of how to setup cloud-controller-manger as
# a Daemonset in your cluster.
# It assumes that your masters can run pods and has the role
# node-role.kubernetes.io/master
# Note that this Daemonset will not work straight out of the box
# for your cloud, this is
# meant to be a guideline.

---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: cloud-controller-manager
```

```

namespace: kube-system
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: system:cloud-controller-manager
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
- kind: ServiceAccount
  name: cloud-controller-manager
  namespace: kube-system
---
apiVersion: apps/v1
kind: DaemonSet
metadata:
  labels:
    k8s-app: cloud-controller-manager
    name: cloud-controller-manager
    namespace: kube-system
spec:
  selector:
    matchLabels:
      k8s-app: cloud-controller-manager
  template:
    metadata:
      labels:
        k8s-app: cloud-controller-manager
    spec:
      serviceAccountName: cloud-controller-manager
      containers:
        - name: cloud-controller-manager
          # for in-tree providers we use k8s.gcr.io/cloud-controller-manager
          # this can be replaced with any other image for out-of-tree providers
          image: k8s.gcr.io/cloud-controller-manager:v1.8.0
          command:
            - /usr/local/bin/cloud-controller-manager
            - --cloud-provider=[YOUR_CLOUD_PROVIDER] # Add your own cloud provider here!
              - --leader-elect=true
              - --use-service-account-credentials
              # these flags will vary for every cloud provider
              - --allocate-node-cidrs=true
              - --configure-cloud-routes=true
              - --cluster-cidr=172.17.0.0/16
      tolerations:
        # this is required so CCM can bootstrap itself
        - key: node.cloudprovider.kubernetes.io/uninitialized

```

```

    value: "true"
    effect: NoSchedule
  # this is to have the daemonset runnable on master nodes
  # the taint may vary depending on your cluster setup
  - key: node-role.kubernetes.io/master
    effect: NoSchedule
  # this is to restrict CCM to only run on master nodes
  # the node selector may vary depending on your cluster
  setup
    nodeSelector:
      node-role.kubernetes.io/master: ""

```

## **Limitations**

*Running cloud controller manager comes with a few possible limitations. Although these limitations are being addressed in upcoming releases, it's important that you are aware of these limitations for production workloads.*

### **Support for Volumes**

*Cloud controller manager does not implement any of the volume controllers found in kube-controller-manager as the volume integrations also require coordination with kubelets. As we evolve CSI (container storage interface) and add stronger support for flex volume plugins, necessary support will be added to cloud controller manager so that clouds can fully integrate with volumes. Learn more about out-of-tree CSI volume plugins [here](#).*

### **Scalability**

*The cloud-controller-manager queries your cloud provider's APIs to retrieve information for all nodes. For very large clusters, consider possible bottlenecks such as resource requirements and API rate limiting.*

### **Chicken and Egg**

*The goal of the cloud controller manager project is to decouple development of cloud features from the core Kubernetes project. Unfortunately, many aspects of the Kubernetes project has assumptions that cloud provider features are tightly integrated into the project. As a result, adopting this new architecture can create several situations where a request is being made for information from a cloud provider, but the cloud controller manager may not be able to return that information without the original request being complete.*

*A good example of this is the TLS bootstrapping feature in the Kubelet. TLS bootstrapping assumes that the Kubelet has the ability to ask the cloud provider (or a local metadata service) for all its address types (private, public, etc) but cloud controller manager cannot set a node's address types without being initialized in the first place which requires that the kubelet has TLS certificates to communicate with the apiserver.*

*As this initiative evolves, changes will be made to address these issues in upcoming releases.*

## **What's next**

*To build and develop your own cloud controller manager, read [Developing Cloud Controller Manager](#).*

## **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](#).*

*Last modified October 12, 2020 at 4:42 PM PST: [Fix a markup error](#).  
(9b55db028)*

[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Administration](#)
  - [Requirements](#)
  - [Running cloud-controller-manager](#)
- [Examples](#)
- [Limitations](#)
  - [Support for Volumes](#)
  - [Scalability](#)
  - [Chicken and Egg](#)
- [What's next](#)

# **Configure Out of Resource Handling**

*This page explains how to configure out of resource handling with kubelet.*

*The kubelet needs to preserve node stability when available compute resources are low. This is especially important when dealing with incompressible compute resources, such as memory or disk space. If such resources are exhausted, nodes become unstable.*

## **Eviction Signals**

*The kubelet supports eviction decisions based on the signals described in the following table. The value of each signal is described in the Description column, which is based on the kubelet summary API.*

<b>Eviction Signal</b>	<b>Description</b>
memory.available	memory.available := node.status.capacity[memory] - node.stats.memory.workingSet
nodefs.available	nodefs.available := node.stats.fs.available
nodefs.inodesFree	nodefs.inodesFree := node.stats.fs.inodesFree
imagefs.available	imagefs.available := node.stats.runtime.imagefs.available
imagefs.inodesFree	imagefs.inodesFree := node.stats.runtime.imagefs.inodesFree
pid.available	pid.available := node.stats.rlimit.maxpid - node.stats.rlimit.curproc

*Each of the above signals supports either a literal or percentage based value. The percentage based value is calculated relative to the total capacity associated with each signal.*

*The value for `memory.available` is derived from the `cgroupfs` instead of tools like `free -m`. This is important because `free -m` does not work in a container, and if users use the [node allocatable](#) feature, out of resource decisions are made local to the end user Pod part of the cgroup hierarchy as well as the root node. This [script](#) reproduces the same set of steps that the `kubelet` performs to calculate `memory.available`. The `kubelet` excludes `inactive_file` (i.e. # of bytes of file-backed memory on inactive LRU list) from its calculation as it assumes that memory is reclaimable under pressure.*

*kubelet supports only two filesystem partitions.*

1. The `nodefs` filesystem that `kubelet` uses for volumes, daemon logs, etc.
2. The `imagefs` filesystem that container runtimes uses for storing images and container writable layers.

*`imagefs` is optional. `kubelet` auto-discovers these filesystems using `cAdvisor`. `kubelet` does not care about any other filesystems. Any other types of configurations are not currently supported by the `kubelet`. For example, it is not OK to store volumes and logs in a dedicated filesystem.*

*In future releases, the `kubelet` will deprecate the existing [garbage collection](#) support in favor of eviction in response to disk pressure.*

## **Eviction Thresholds**

*The `kubelet` supports the ability to specify eviction thresholds that trigger the `kubelet` to reclaim resources.*

*Each threshold has the following form:*

*[eviction-signal][operator][quantity]*

*where:*

- *`eviction-signal` is an eviction signal token as defined in the previous table.*
- *`operator` is the desired relational operator, such as < (less than).*

- *quantity* is the eviction threshold quantity, such as `1Gi`. These tokens must match the quantity representation used by Kubernetes. An eviction threshold can also be expressed as a percentage using the `%` token.

For example, if a node has `10Gi` of total memory and you want trigger eviction if the available memory falls below `1Gi`, you can define the eviction threshold as either `memory.available<10%` or `memory.available<1Gi`. You cannot use both.

## **Soft Eviction Thresholds**

A soft eviction threshold pairs an eviction threshold with a required administrator-specified grace period. No action is taken by the `kubelet` to reclaim resources associated with the eviction signal until that grace period has been exceeded. If no grace period is provided, the `kubelet` returns an error on startup.

In addition, if a soft eviction threshold has been met, an operator can specify a maximum allowed Pod termination grace period to use when evicting pods from the node. If specified, the `kubelet` uses the lesser value among the `pod.Spec.TerminationGracePeriodSeconds` and the max allowed grace period. If not specified, the `kubelet` kills Pods immediately with no graceful termination.

To configure soft eviction thresholds, the following flags are supported:

- `eviction-soft` describes 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` describes 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.
- `eviction-max-pod-grace-period` describes the maximum allowed grace period (in seconds) to use when terminating pods in response to a soft eviction threshold being met.

## **Hard Eviction Thresholds**

A hard eviction threshold has no grace period, and if observed, the `kubelet` will take immediate action to reclaim the associated starved resource. If a hard eviction threshold is met, the `kubelet` kills the Pod immediately with no graceful termination.

To configure hard eviction thresholds, the following flag is supported:

- `eviction-hard` describes a set of eviction thresholds (e.g. `memory.available<1Gi`) that if met would trigger a Pod eviction.

The `kubelet` has the following default hard eviction threshold:

- `memory.available<100Mi`

- `nodefs.available<10%`
- `nodefs.inodesFree<5%`
- `imagefs.available<15%`

## **Eviction Monitoring Interval**

The kubelet evaluates eviction thresholds per its configured housekeeping interval.

- `housekeeping-interval` is the interval between container housekeepings which defaults to 10s.

## **Node Conditions**

The kubelet maps one or more eviction signals to a corresponding node condition.

If a hard eviction threshold has been met, or a soft eviction threshold has been met independent of its associated grace period, the kubelet reports a condition that reflects the node is under pressure.

The following node conditions are defined that correspond to the specified eviction signal.

<b>Node Condition</b>	<b>Eviction Signal</b>	<b>Description</b>
MemoryPressure	<code>memory.available</code>	Available memory on the node satisfied eviction threshold
DiskPressure	<code>nodefs.available, nodefs.inodesFree, imagefs.available, or imagefs.inodesFree</code>	Available disk space and inodes on either the node root filesystem image filesystem has satisfied an eviction threshold

The kubelet continues to report node status updates at the frequency specified by `--node-status-update-frequency` which defaults to 10s.

## **Oscillation of node conditions**

*If a node is oscillating above and below a soft eviction threshold, but not exceeding its associated grace period, it would cause the corresponding node condition to constantly oscillate between true and false, and could cause poor scheduling decisions as a consequence.*

*To protect against this oscillation, the following flag is defined to control how long the kubelet must wait before transitioning out of a pressure condition.*

- *eviction-pressure-transition-period* is the duration for which the kubelet has to wait before transitioning out of an eviction pressure condition.

*The kubelet would ensure that it has not observed an eviction threshold being met for the specified pressure condition for the period specified before toggling the condition back to false.*

## **Reclaiming node level resources**

*If an eviction threshold has been met and the grace period has passed, the kubelet initiates the process of reclaiming the pressured resource until it has observed the signal has gone below its defined threshold.*

*The kubelet attempts to reclaim node level resources prior to evicting end-user Pods. If disk pressure is observed, the kubelet reclaims node level resources differently if the machine has a dedicated `imagefs` configured for the container runtime.*

### **With `imagefs`**

*If `nodefs` filesystem has met eviction thresholds, kubelet frees up disk space by deleting the dead Pods and their containers.*

*If `imagefs` filesystem has met eviction thresholds, kubelet frees up disk space by deleting all unused images.*

### **Without `imagefs`**

*If `nodefs` filesystem has met eviction thresholds, kubelet frees up disk space in the following order:*

1. Delete dead Pods and their containers
2. Delete all unused images

## **Evicting end-user Pods**

*If the kubelet is unable to reclaim sufficient resource on the node, kubelet begins evicting Pods.*

*The kubelet ranks Pods for eviction first by whether or not their usage of the starved resource exceeds requests, then by [Priority](#), and then by the consumption of the starved compute resource relative to the Pods' scheduling requests.*

*As a result, kubelet ranks and evicts Pods in the following order:*

- *BestEffort or Burstable Pods whose usage of a starved resource exceeds its request. Such pods are ranked by Priority, and then usage above request.*
- *Guaranteed pods and Burstable pods whose usage is beneath requests are evicted last. Guaranteed Pods are guaranteed only when requests and limits are specified for all the containers and they are equal. Such pods are guaranteed to never be evicted because of another Pod's resource consumption. If a system daemon (such as kubelet, docker, and journald) is consuming more resources than were reserved via system-reserved or kube-reserved allocations, and the node only has Guaranteed or Burstable Pods using less than requests remaining, then the node must choose to evict such a Pod in order to preserve node stability and to limit the impact of the unexpected consumption to other Pods. In this case, it will choose to evict pods of Lowest Priority first.*

*If necessary, kubelet evicts Pods one at a time to reclaim disk when DiskPressure is encountered. If the kubelet is responding to inode starvation, it reclaims inodes by evicting Pods with the lowest quality of service first. If the kubelet is responding to lack of available disk, it ranks Pods within a quality of service that consumes the largest amount of disk and kills those first.*

### **With imagefs**

*If nodefs is triggering evictions, kubelet sorts Pods based on the usage on nodefs*

- local volumes + logs of all its containers.

*If imagefs is triggering evictions, kubelet sorts Pods based on the writable layer usage of all its containers.*

### **Without imagefs**

*If nodefs is triggering evictions, kubelet sorts Pods based on their total disk usage*

- local volumes + logs & writable layer of all its containers.

### **Minimum eviction reclaim**

*In certain scenarios, eviction of Pods could result in reclamation of small amount of resources. This can result in kubelet hitting eviction thresholds in repeated successions. In addition to that, eviction of resources like disk, is time consuming.*

To mitigate these issues, kubelet can have a per-resource *minimum-reclaim*. Whenever kubelet observes resource pressure, kubelet attempts to reclaim at least *minimum-reclaim* amount of resource below the configured eviction threshold.

For example, with the following configuration:

```
--eviction-
hard=memory.available<500Mi,nodefs.available<1Gi,imagefs.available<100Gi
--eviction-minimum-
reclaim="memory.available=0Mi,nodefs.available=500Mi,imagefs.available=2Gi"
```

If an eviction threshold is triggered for *memory.available*, the kubelet works to ensure that *memory.available* is at least 500Mi. For *nodefs.available*, the kubelet works to ensure that *nodefs.available* is at least 1.5Gi, and for *imagefs.available* it works to ensure that *imagefs.available* is at least 102Gi before no longer reporting pressure on their associated resources.

The default *eviction-minimum-reclaim* is 0 for all resources.

## Scheduler

The node reports a condition when a compute resource is under pressure. The scheduler views that condition as a signal to dissuade placing additional pods on the node.

Node Condition	Scheduler Behavior
MemoryPressure	No new BestEffort Pods are scheduled to the node.
DiskPressure	No new Pods are scheduled to the node.

## Node OOM Behavior

If the node experiences a system OOM (out of memory) event prior to the kubelet being able to reclaim memory, the node depends on the [oom\\_killer](#) to respond.

The kubelet sets a *oom\_score\_adj* value for each container based on the quality of service for the Pod.

Quality of Service	oom_score_adj
Guaranteed	-998
BestEffort	1000
Burstable	$\min(\max(2, 1000 - (1000 * \text{memoryRequestBytes}) / \text{machineMemoryCapacityBytes}), 999)$

If the kubelet is unable to reclaim memory prior to a node experiencing system OOM, the *oom\_killer* calculates an *oom\_score* based on the

*percentage of memory it's using on the node, and then add the `oom_score_adj` to get an effective `oom_score` for the container, and then kills the container with the highest score.*

*The intended behavior should be that containers with the lowest quality of service that are consuming the largest amount of memory relative to the scheduling request should be killed first in order to reclaim memory.*

*Unlike Pod eviction, if a Pod container is OOM killed, it may be restarted by the `kubelet` based on its `RestartPolicy`.*

## **Best Practices**

*The following sections describe best practices for out of resource handling.*

### **Schedulable resources and eviction policies**

*Consider the following scenario:*

- *Node memory capacity: 10Gi*
- *Operator wants to reserve 10% of memory capacity for system daemons (kernel, `kubelet`, etc.)*
- *Operator wants to evict Pods at 95% memory utilization to reduce incidence of system OOM.*

*To facilitate this scenario, the `kubelet` would be launched as follows:*

```
--eviction-hard=memory.available<500Mi  
--system-reserved=memory=1.5Gi
```

*Implicit in this configuration is the understanding that "System reserved" should include the amount of memory covered by the eviction threshold.*

*To reach that capacity, either some Pod is using more than its request, or the system is using more than 1.5Gi - 500Mi = 1Gi.*

*This configuration ensures that the scheduler does not place Pods on a node that immediately induce memory pressure and trigger eviction assuming those Pods use less than their configured request.*

### **DaemonSet**

*As Priority is a key factor in the eviction strategy, if you do not want pods belonging to a DaemonSet to be evicted, specify a sufficiently high `priorityClass` in the pod spec template. If you want pods belonging to a DaemonSet to run only if there are sufficient resources, specify a lower or default `priorityClass`.*

# **Deprecation of existing feature flags to reclaim disk**

*kubelet has been freeing up disk space on demand to keep the node stable.*

*As disk based eviction matures, the following kubelet flags are marked for deprecation in favor of the simpler configuration supported around eviction.*

Existing Flag	New Flag
--image-gc-high-threshold	--eviction-hard or eviction-soft
--image-gc-low-threshold	--eviction-minimum-reclaim
--maximum-dead-containers	deprecated
--maximum-dead-containers-per-container	deprecated
--minimum-container-ttl-duration	deprecated
--low-diskspace-threshold-mb	--eviction-hard or eviction-soft
--outofdisk-transition-frequency	--eviction-pressure-transition-period

## **Known issues**

*The following sections describe known issues related to out of resource handling.*

### ***kubelet may not observe memory pressure right away***

*The kubelet currently polls cAdvisor to collect memory usage stats at a regular interval. If memory usage increases within that window rapidly, the kubelet may not observe MemoryPressure fast enough, and the OOMKiller will still be invoked. We intend to integrate with the memcg notification API in a future release to reduce this latency, and instead have the kernel tell us when a threshold has been crossed immediately.*

*If you are not trying to achieve extreme utilization, but a sensible measure of overcommit, a viable workaround for this issue is to set eviction thresholds at approximately 75% capacity. This increases the ability of this feature to prevent system OOMs, and promote eviction of workloads so cluster state can rebalance.*

### ***kubelet may evict more Pods than needed***

*The Pod eviction may evict more Pods than needed due to stats collection timing gap. This can be mitigated by adding the ability to get root container stats on an on-demand basis (<https://github.com/google/cadvisor/issues/1247>) in the future.*

## ***active\_file memory is not considered as available memory***

*On Linux, the kernel tracks the number of bytes of file-backed memory on active LRU list as the `active_file` statistic. The kubelet treats `active_file` memory areas as not reclaimable. For workloads that make intensive use of block-backed local storage, including ephemeral local storage, kernel-level caches of file and block data means that many recently accessed cache pages are likely to be counted as `active_file`. If enough of these kernel block buffers are on the active LRU list, the kubelet is liable to observe this as high resource use and taint the node as experiencing memory pressure - triggering Pod eviction.*

*For more details, see <https://github.com/kubernetes/kubernetes/issues/43916>*

*You can work around that behavior by setting the memory limit and memory request the same for containers likely to perform intensive I/O activity. You will need to estimate or measure an optimal memory limit value for that container.*

## **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](#).

Last modified December 08, 2020 at 10:58 PM PST: [add `pid.available` to the eviction signals list \(d1dc73cb3\)](#)

[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Eviction Signals](#)
  - [Eviction Thresholds](#)
  - [Eviction Monitoring Interval](#)
  - [Node Conditions](#)
  - [Oscillation of node conditions](#)
  - [Reclaiming node level resources](#)
  - [Evicting end-user Pods](#)
  - [Minimum eviction reclaim](#)
  - [Scheduler](#)
- [Node OOM Behavior](#)
- [Best Practices](#)
  - [Schedulable resources and eviction policies](#)
  - [DaemonSet](#)
- [Deprecation of existing feature flags to reclaim disk](#)
- [Known issues](#)
  - [kubelet may not observe memory pressure right away](#)
  - [kubelet may evict more Pods than needed](#)
  - [active\\_file memory is not considered as available memory](#)

# Configure Quotas for API Objects

This page shows how to configure quotas for API objects, including `PersistentVolumeClaims` and `Services`. A quota restricts the number of objects, of a particular type, that can be created in a namespace. You specify quotas in a [ResourceQuota](#) object.

## Before you begin

You need to have a Kubernetes cluster, and the `kubectl` command-line tool must be configured to communicate with your cluster. If you do not already have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:

- [Katacoda](#)
- [Play with Kubernetes](#)

To check the version, enter `kubectl version`.

## Create a namespace

Create a namespace so that the resources you create in this exercise are isolated from the rest of your cluster.

```
kubectl create namespace quota-object-example
```

## Create a ResourceQuota

Here is the configuration file for a `ResourceQuota` object:

[admin/resource/quota-objects.yaml](#)



```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: object-quota-demo
spec:
  hard:
    persistentvolumeclaims: "1"
    services.loadbalancers: "2"
    services.nodeports: "0"
```

Create the `ResourceQuota`:

```
kubectl apply -f https://k8s.io/examples/admin/resource/quota-objects.yaml --namespace=quota-object-example
```

*View detailed information about the ResourceQuota:*

```
kubectl get resourcequota object-quota-demo --namespace=quota-object-example --output=yaml
```

*The output shows that in the quota-object-example namespace, there can be at most one PersistentVolumeClaim, at most two Services of type LoadBalancer, and no Services of type NodePort.*

```
status:  
  hard:  
    persistentvolumeclaims: "1"  
    services.loadbalancers: "2"  
    services.nodeports: "0"  
  used:  
    persistentvolumeclaims: "0"  
    services.loadbalancers: "0"  
    services.nodeports: "0"
```

## Create a PersistentVolumeClaim

*Here is the configuration file for a PersistentVolumeClaim object:*

[admin/resource/quota-objects-pvc.yaml](#)



```
apiVersion: v1  
kind: PersistentVolumeClaim  
metadata:  
  name: pvc-quota-demo  
spec:  
  storageClassName: manual  
  accessModes:  
    - ReadWriteOnce  
  resources:  
    requests:  
      storage: 3Gi
```

*Create the PersistentVolumeClaim:*

```
kubectl apply -f https://k8s.io/examples/admin/resource/quota-objects-pvc.yaml --namespace=quota-object-example
```

Verify that the PersistentVolumeClaim was created:

```
kubectl get persistentvolumeclaims --namespace=quota-object-example
```

The output shows that the PersistentVolumeClaim exists and has status Pending:

NAME	STATUS
pvc-quota-demo	Pending

## Attempt to create a second PersistentVolumeClaim

Here is the configuration file for a second PersistentVolumeClaim:

[admin/resource/quota-objects-pvc-2.yaml](https://k8s.io/examples/admin/resource/quota-objects-pvc-2.yaml)

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-quota-demo-2
spec:
  storageClassName: manual
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 4Gi
```

Attempt to create the second PersistentVolumeClaim:

```
kubectl apply -f https://k8s.io/examples/admin/resource/quota-objects-pvc-2.yaml --namespace=quota-object-example
```

The output shows that the second PersistentVolumeClaim was not created, because it would have exceeded the quota for the namespace.

```
persistentvolumeclaims "pvc-quota-demo-2" is forbidden:
exceeded quota: object-quota-demo, requested:
persistentvolumeclaims=1,
used: persistentvolumeclaims=1, limited: persistentvolumeclaims=1
```

# Notes

These are the strings used to identify API resources that can be constrained by quotas:

<b>String</b>	<b>API Object</b>
"pods"	<i>Pod</i>
"services"	<i>Service</i>
"replicationcontrollers"	<i>ReplicationController</i>
"resourcequotas"	<i>ResourceQuota</i>
"secrets"	<i>Secret</i>
"configmaps"	<i>ConfigMap</i>
"persistentvolumeclaims"	<i>PersistentVolumeClaim</i>
"services.nodeports"	<i>Service of type NodePort</i>
"services.loadbalancers"	<i>Service of type LoadBalancer</i>

## Clean up

Delete your namespace:

```
kubectl delete namespace quota-object-example
```

## What's next

### For cluster administrators

- [Configure Default Memory Requests and Limits for a Namespace](#)
- [Configure Default CPU Requests and Limits for a Namespace](#)
- [Configure Minimum and Maximum Memory Constraints for a Namespace](#)
- [Configure Minimum and Maximum CPU Constraints for a Namespace](#)
- [Configure Memory and CPU Quotas for a Namespace](#)
- [Configure a Pod Quota for a Namespace](#)

## **For app developers**

- [Assign Memory Resources to Containers and Pods](#)
- [Assign CPU Resources to Containers and Pods](#)
- [Configure Quality of Service for Pods](#)

## **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](#).

Last modified July 29, 2020 at 9:40 PM PST: [Fix broken links to pages under /en/docs/tasks/administer-cluster/manage-resources/ \(36d9239fb\)](#)  
[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Create a namespace](#)
- [Create a ResourceQuota](#)
- [Create a PersistentVolumeClaim](#)
- [Attempt to create a second PersistentVolumeClaim](#)
- [Notes](#)
- [Clean up](#)
- [What's next](#)
  - [For cluster administrators](#)
  - [For app developers](#)

# **Control CPU Management Policies on the Node**

**FEATURE STATE:** Kubernetes v1.12 [beta]

Kubernetes keeps many aspects of how pods execute on nodes abstracted from the user. This is by design. However, some workloads require stronger guarantees in terms of latency and/or performance in order to operate acceptably. The kubelet provides methods to enable more complex workload placement policies while keeping the abstraction free from explicit placement directives.

## **Before you begin**

You need to have a Kubernetes cluster, and the `kubectl` command-line tool must be configured to communicate with your cluster. If you do not already have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:

- [Katacoda](#)
- [Play with Kubernetes](#)

To check the version, enter `kubectl version`.

## CPU Management Policies

By default, the `kubelet` uses [CFS quota](#) to enforce pod CPU limits. When the node runs many CPU-bound pods, the workload can move to different CPU cores depending on whether the pod is throttled and which CPU cores are available at scheduling time. Many workloads are not sensitive to this migration and thus work fine without any intervention.

However, in workloads where CPU cache affinity and scheduling latency significantly affect workload performance, the `kubelet` allows alternative CPU management policies to determine some placement preferences on the node.

### Configuration

The CPU Manager policy is set with the `--cpu-manager-policy` `kubelet` option. There are two supported policies:

- [none](#): the default policy.
- [static](#): allows pods with certain resource characteristics to be granted increased CPU affinity and exclusivity on the node.

The CPU manager periodically writes resource updates through the CRI in order to reconcile in-memory CPU assignments with cgroupfs. The reconcile frequency is set through a new Kubelet configuration value `--cpu-manager-reconcile-period`. If not specified, it defaults to the same duration as `--node-status-update-frequency`.

### None policy

The `none` policy explicitly enables the existing default CPU affinity scheme, providing no affinity beyond what the OS scheduler does automatically. Limits on CPU usage for [Guaranteed pods](#) are enforced using CFS quota.

## **Static policy**

The static policy allows containers in Guaranteed pods with integer CPU requests access to exclusive CPUs on the node. This exclusivity is enforced using the [cpuset cgroup controller](#).

**Note:** System services such as the container runtime and the kubelet itself can continue to run on these exclusive CPUs. ▶ The exclusivity only extends to other pods.

**Note:** CPU Manager doesn't support offline and online of CPUs at runtime. Also, if the set of online CPUs changes on the node, the node must be drained and CPU manager manually reset by deleting the state file `cpu_manager_state` in the kubelet root directory.

This policy manages a shared pool of CPUs that initially contains all CPUs in the node. The amount of exclusively allocatable CPUs is equal to the total number of CPUs in the node minus any CPU reservations by the kubelet --kube-reserved or --system-reserved options. From 1.17, the CPU reservation list can be specified explicitly by kubelet --reserved-cpus option. The explicit CPU list specified by --reserved-cpus takes precedence over the CPU reservation specified by --kube-reserved and --system-reserved. CPUs reserved by these options are taken, in integer quantity, from the initial shared pool in ascending order by physical core ID. ▶ This shared pool is the set of CPUs on which any containers in BestEffort and Burstable pods run. Containers in Guaranteed pods with fractional CPU requests also run on CPUs in the shared pool. Only containers that are both part of a Guaranteed pod and have integer CPU requests are assigned exclusive CPUs.

**Note:** The kubelet requires a CPU reservation greater than zero be made using either --kube-reserved and/or --system-reserved or --reserved-cpus when the static policy is enabled. This is because zero CPU reservation would allow the shared pool to become empty.

As Guaranteed pods whose containers fit the requirements for being statically assigned are scheduled to the node, CPUs are removed from the shared pool and placed in the cpuset for the container. CFS quota is not used to bound the CPU usage of these containers as their usage is bound by the scheduling domain itself. In others words, the number of CPUs in the container cpuset is equal to the integer CPU limit specified in the pod spec. ▶ This static assignment increases CPU affinity and decreases context switches due to throttling for the CPU-bound workload.

Consider the containers in the following pod specs:

```
spec:  
  containers:  
    - name: nginx  
      image: nginx
```

This pod runs in the `BestEffort` QoS class because no resource requests or limits are specified. It runs in the shared pool.

```
spec:  
  containers:  
    - name: nginx  
      image: nginx  
      resources:  
        limits:  
          memory: "200Mi"  
        requests:  
          memory: "100Mi"
```

This pod runs in the `Burstable` QoS class because resource requests do not equal limits and the `cpu` quantity is not specified. It runs in the shared pool.

```
spec:  
  containers:  
    - name: nginx  
      image: nginx  
      resources:  
        limits:  
          memory: "200Mi"  
          cpu: "2"  
        requests:  
          memory: "100Mi"  
          cpu: "1"
```

This pod runs in the `Burstable` QoS class because resource requests do not equal limits. It runs in the shared pool.

```
spec:  
  containers:  
    - name: nginx  
      image: nginx  
      resources:  
        limits:  
          memory: "200Mi"  
          cpu: "2"
```

```
  requests:  
    memory: "200Mi"  
    cpu: "2"
```

*This pod runs in the Guaranteed QoS class because requests are equal to limits. And the container's resource limit for the CPU resource is an integer greater than or equal to one. The nginx container is granted 2 exclusive CPUs.*

```
spec:  
  containers:  
  - name: nginx  
    image: nginx  
    resources:  
      limits:  
        memory: "200Mi"  
        cpu: "1.5"  
      requests:  
        memory: "200Mi"  
        cpu: "1.5"
```

*This pod runs in the Guaranteed QoS class because requests are equal to limits. But the container's resource limit for the CPU resource is a fraction. It runs in the shared pool.*

```
spec:  
  containers:  
  - name: nginx  
    image: nginx  
    resources:  
      limits:  
        memory: "200Mi"  
        cpu: "2"
```

*This pod runs in the Guaranteed QoS class because only limits are specified and requests are set equal to limits when not explicitly specified. And the container's resource limit for the CPU resource is an integer greater than or equal to one. The nginx container is granted 2 exclusive CPUs.*

## 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](#).

Last modified June 23, 2020 at 2:56 PM PST: [Remove extra space in documentation \(a0bfa31a4\)](#)

[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [CPU Management Policies](#)
  - [Configuration](#)
  - [None policy](#)
  - [Static policy](#)

# **Control Topology Management Policies on a node**

**FEATURE STATE:** Kubernetes v1.18 [beta]

An increasing number of systems leverage a combination of CPUs and hardware accelerators to support latency-critical execution and high-throughput parallel computation. These include workloads in fields such as telecommunications, scientific computing, machine learning, financial services and data analytics. Such hybrid systems comprise a high performance environment.

In order to extract the best performance, optimizations related to CPU isolation, memory and device locality are required. However, in Kubernetes, these optimizations are handled by a disjoint set of components.

*Topology Manager* is a Kubelet component that aims to co-ordinate the set of components that are responsible for these optimizations.

## **Before you begin**

You need to have a Kubernetes cluster, and the `kubectl` command-line tool must be configured to communicate with your cluster. If you do not already have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:

- [Katacoda](#)
- [Play with Kubernetes](#)

Your Kubernetes server must be at or later than version v1.18. To check the version, enter `kubectl version`.

## **How Topology Manager Works**

*Prior to the introduction of Topology Manager, the CPU and Device Manager in Kubernetes make resource allocation decisions independently of each other. This can result in undesirable allocations on multiple-socketed systems, performance/latency sensitive applications will suffer due to these undesirable allocations. Undesirable in this case meaning for example, CPUs and devices being allocated from different NUMA Nodes thus, incurring additional latency.*

*The Topology Manager is a Kubelet component, which acts as a source of truth so that other Kubelet components can make topology aligned resource allocation choices.*

*The Topology Manager provides an interface for components, called Hint Providers, to send and receive topology information. Topology Manager has a set of node level policies which are explained below.*

*The Topology manager receives Topology information from the Hint Providers as a bitmask denoting NUMA Nodes available and a preferred allocation indication. The Topology Manager policies perform a set of operations on the hints provided and converge on the hint determined by the policy to give the optimal result, if an undesirable hint is stored the preferred field for the hint will be set to false. In the current policies preferred is the narrowest preferred mask. The selected hint is stored as part of the Topology Manager. Depending on the policy configured the pod can be accepted or rejected from the node based on the selected hint. The hint is then stored in the Topology Manager for use by the Hint Providers when making the resource allocation decisions.*

## **Enable the Topology Manager feature**

*Support for the Topology Manager requires `TopologyManager` [feature gate](#) to be enabled. It is enabled by default starting with Kubernetes 1.18.*

## **Topology Manager Scopes and Policies**

*The Topology Manager currently:*

- Aligns Pods of all QoS classes.
- Aligns the requested resources that Hint Provider provides topology hints for.

*If these conditions are met, the Topology Manager will align the requested resources.*

*In order to customise how this alignment is carried out, the Topology Manager provides two distinct knobs: `scope` and `policy`.*

*The `scope` defines the granularity at which you would like resource alignment to be performed (e.g. at the `pod` or `container` level). And the `policy` defines the actual strategy used to carry out the alignment (e.g. `best-effort`, `restricted`, `single-numa-node`, etc.).*

*Details on the various `scopes` and `policies` available today can be found below.*

**Note:** To align CPU resources with other requested resources in a Pod Spec, the CPU Manager should be enabled and proper CPU Manager policy should be configured on a Node. See [control CPU Management Policies](#).

## **Topology Manager Scopes**

*The Topology Manager can deal with the alignment of resources in a couple of distinct scopes:*

- `container` (default)
- `pod`

*Either option can be selected at a time of the kubelet startup, with `--topology-manager-scope` flag.*

### **container scope**

*The `container` scope is used by default.*

*Within this scope, the Topology Manager performs a number of sequential resource alignments, i.e., for each container (in a pod) a separate alignment is computed. In other words, there is no notion of grouping the containers to a specific set of NUMA nodes, for this particular scope. In effect, the Topology Manager performs an arbitrary alignment of individual containers to NUMA nodes.*

*The notion of grouping the containers was endorsed and implemented on purpose in the following scope, for example the `pod` scope.*

## **pod scope**

*To select the pod scope, start the kubelet with the command line option --topology-manager-scope=pod.*

*This scope allows for grouping all containers in a pod to a common set of NUMA nodes. That is, the Topology Manager treats a pod as a whole and attempts to allocate the entire pod (all containers) to either a single NUMA node or a common set of NUMA nodes. The following examples illustrate the alignments produced by the Topology Manager on different occasions:*

- *all containers can be and are allocated to a single NUMA node;*
- *all containers can be and are allocated to a shared set of NUMA nodes.*

*The total amount of particular resource demanded for the entire pod is calculated according to [effective requests/limits](#) formula, and thus, this total value is equal to the maximum of:*

- *the sum of all app container requests,*
- *the maximum of init container requests, for a resource.*

*Using the pod scope in tandem with single-numa-node Topology Manager policy is specifically valuable for workloads that are latency sensitive or for high-throughput applications that perform IPC. By combining both options, you are able to place all containers in a pod onto a single NUMA node; hence, the inter-NUMA communication overhead can be eliminated for that pod.*

*In the case of single-numa-node policy, a pod is accepted only if a suitable set of NUMA nodes is present among possible allocations. Reconsider the example above:*

- *a set containing only a single NUMA node - it leads to pod being admitted,*
- *whereas a set containing more NUMA nodes - it results in pod rejection (because instead of one NUMA node, two or more NUMA nodes are required to satisfy the allocation).*

*To recap, Topology Manager first computes a set of NUMA nodes and then tests it against Topology Manager policy, which either leads to the rejection or admission of the pod.*

## **Topology Manager Policies**

*Topology Manager supports four allocation policies. You can set a policy via a Kubelet flag, `--topology-manager-policy`. There are four supported policies:*

- *none (default)*
- *best-effort*
- *restricted*
- *single-numa-node*

**Note:** If Topology Manager is configured with the **pod** scope, the container, which is considered by the policy, is reflecting requirements of the entire pod, and thus each container from the pod will result with **the same** topology alignment decision.

## **none policy**

*This is the default policy and does not perform any topology alignment.*

## **best-effort policy**

*For each container in a Pod, the kubelet, with best-effort topology management policy, calls each Hint Provider to discover their resource availability. Using this information, the Topology Manager stores the preferred NUMA Node affinity for that container. If the affinity is not preferred, Topology Manager will store this and admit the pod to the node anyway.*

*The Hint Providers can then use this information when making the resource allocation decision.*

## **restricted policy**

*For each container in a Pod, the kubelet, with restricted topology management policy, calls each Hint Provider to discover their resource availability. Using this information, the Topology Manager stores the preferred NUMA Node affinity for that container. If the affinity is not preferred, Topology Manager will reject this pod from the node. This will result in a pod in a Terminated state with a pod admission failure.*

*Once the pod is in a Terminated state, the Kubernetes scheduler will **not** attempt to reschedule the pod. It is recommended to use a ReplicaSet or Deployment to trigger a redeploy of the pod. An external control loop could*

be also implemented to trigger a redeployment of pods that have the *Topology Affinity* error.

If the pod is admitted, the Hint Providers can then use this information when making the resource allocation decision.

## **single-numa-node policy**

For each container in a Pod, the kubelet, with *single-numa-node* topology management policy, calls each Hint Provider to discover their resource availability. Using this information, the Topology Manager determines if a single NUMA Node affinity is possible. If it is, Topology Manager will store this and the Hint Providers can then use this information when making the resource allocation decision. If, however, this is not possible then the Topology Manager will reject the pod from the node. This will result in a pod in a *Terminated* state with a pod admission failure.

Once the pod is in a *Terminated* state, the Kubernetes scheduler will **not** attempt to reschedule the pod. It is recommended to use a Deployment with replicas to trigger a redeploy of the Pod. An external control loop could be also implemented to trigger a redeployment of pods that have the *Topology Affinity* error.

## **Pod Interactions with Topology Manager Policies**

Consider the containers in the following pod specs:

```
spec:  
  containers:  
    - name: nginx  
      image: nginx
```

This pod runs in the *BestEffort* QoS class because no resource requests or limits are specified.

```
spec:  
  containers:  
    - name: nginx  
      image: nginx  
      resources:  
        limits:  
          memory: "200Mi"  
        requests:  
          memory: "100Mi"
```

*This pod runs in the Burstable QoS class because requests are less than limits.*

*If the selected policy is anything other than none, Topology Manager would consider these Pod specifications. The Topology Manager would consult the Hint Providers to get topology hints. In the case of the static, the CPU Manager policy would return default topology hint, because these Pods do not have explicitly request CPU resources.*

```
spec:  
  containers:  
    - name: nginx  
      image: nginx  
      resources:  
        limits:  
          memory: "200Mi"  
          cpu: "2"  
          example.com/device: "1"  
        requests:  
          memory: "200Mi"  
          cpu: "2"  
          example.com/device: "1"
```

*This pod with integer CPU request runs in the Guaranteed QoS class because requests are equal to limits.*

```
spec:  
  containers:  
    - name: nginx  
      image: nginx  
      resources:  
        limits:  
          memory: "200Mi"  
          cpu: "300m"  
          example.com/device: "1"  
        requests:  
          memory: "200Mi"  
          cpu: "300m"  
          example.com/device: "1"
```

*This pod with sharing CPU request runs in the Guaranteed QoS class because requests are equal to limits.*

```
spec:  
  containers:  
    - name: nginx  
      image: nginx
```

```
resources:  
  limits:  
    example.com/deviceA: "1"  
    example.com/deviceB: "1"  
  requests:  
    example.com/deviceA: "1"  
    example.com/deviceB: "1"
```

*This pod runs in the BestEffort QoS class because there are no CPU and memory requests.*

*The Topology Manager would consider the above pods. The Topology Manager would consult the Hint Providers, which are CPU and Device Manager to get topology hints for the pods.*

*In the case of the Guaranteed pod with integer CPU request, the static CPU Manager policy would return topology hints relating to the exclusive CPU and the Device Manager would send back hints for the requested device.*

*In the case of the Guaranteed pod with sharing CPU request, the static CPU Manager policy would return default topology hint as there is no exclusive CPU request and the Device Manager would send back hints for the requested device.*

*In the above two cases of the Guaranteed pod, the none CPU Manager policy would return default topology hint.*

*In the case of the BestEffort pod, the static CPU Manager policy would send back the default topology hint as there is no CPU request and the Device Manager would send back the hints for each of the requested devices.*

*Using this information the Topology Manager calculates the optimal hint for the pod and stores this information, which will be used by the Hint Providers when they are making their resource assignments.*

## **Known Limitations**

1. The maximum number of NUMA nodes that Topology Manager allows is 8. With more than 8 NUMA nodes there will be a state explosion when trying to enumerate the possible NUMA affinities and generating their hints.

*The scheduler is not topology-aware, so it is possible to be scheduled on a node and then fail on the node due to the Topology Manager.*

*3. The Device Manager and the CPU Manager are the only components to adopt the Topology Manager's HintProvider interface. This means that NUMA alignment can only be achieved for resources managed by the CPU Manager and the Device Manager. Memory or Hugepages are not considered by the Topology Manager for NUMA alignment.*

## 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](#).

Last modified November 13, 2020 at 5:11 PM PST: [Update Topology Manager documentation to include the scope feature \(425539369\)](#)  
[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [How Topology Manager Works](#)
  - [Enable the Topology Manager feature](#)
- [Topology Manager Scopes and Policies](#)
  - [Topology Manager Scopes](#)
  - [container scope](#)
  - [pod scope](#)
  - [Topology Manager Policies](#)
  - [none policy](#)
  - [best-effort policy](#)
  - [restricted policy](#)
  - [single-numa-node policy](#)
  - [Pod Interactions with Topology Manager Policies](#)
  - [Known Limitations](#)

## Customizing DNS Service

This page explains how to configure your DNS [Pod\(s\)](#) and customize the DNS resolution process in your cluster.

### Before you begin

You need to have a Kubernetes cluster, and the `kubectl` command-line tool must be configured to communicate with your cluster. If you do not already

have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:

- [Katacoda](#)
- [Play with Kubernetes](#)

Your cluster must be running the CoreDNS add-on. [Migrating to CoreDNS](#) explains how to use `kubeadm` to migrate from `kube-dns`.

Your Kubernetes server must be at or later than version v1.12. To check the version, enter `kubectl version`.

## Introduction

DNS is a built-in Kubernetes service launched automatically using the addon manager [cluster add-on](#).

As of Kubernetes v1.12, CoreDNS is the recommended DNS Server, replacing `kube-dns`. If your cluster originally used `kube-dns`, you may still have `kube-dns` deployed rather than CoreDNS.

**Note:** Both the CoreDNS and `kube-dns` Service are named `kube-dns` in the `metadata.name` field.

This is so that there is greater interoperability with workloads that relied on the legacy `kube-dns` Service name to resolve addresses internal to the cluster. Using a Service named `kube-dns` abstracts away the implementation detail of which DNS provider is running behind that common name.

If you are running CoreDNS as a Deployment, it will typically be exposed as a Kubernetes Service with a static IP address. The kubelet passes DNS resolver information to each container with the `--cluster-dns=<dns-service-ip>` flag.

DNS names also need domains. You configure the local domain in the kubelet with the flag `--cluster-domain=<default-local-domain>`.

The DNS server supports forward lookups (A and AAAA records), port lookups (SRV records), reverse IP address lookups (PTR records), and more. For more information, see [DNS for Services and Pods](#).

If a Pod's `dnsPolicy` is set to `default`, it inherits the name resolution configuration from the node that the Pod runs on. The Pod's DNS resolution should behave the same as the node. But see [Known issues](#).

If you don't want this, or if you want a different DNS config for pods, you can use the kubelet's `--resolv-conf` flag. Set this flag to `""` to prevent Pods from inheriting DNS. Set it to a valid file path to specify a file other than `/etc/resolv.conf` for DNS inheritance.

# CoreDNS

CoreDNS is a general-purpose authoritative DNS server that can serve as cluster DNS, complying with the [dns specifications](#).

## CoreDNS ConfigMap options

CoreDNS is a DNS server that is modular and pluggable, and each plugin adds new functionality to CoreDNS. This can be configured by maintaining a [Corefile](#), which is the CoreDNS configuration file. As a cluster administrator, you can modify the [ConfigMap](#) for the CoreDNS Corefile to change how DNS service discovery behaves for that cluster.

In Kubernetes, CoreDNS is installed with the following default Corefile configuration:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: coredns
  namespace: kube-system
data:
  Corefile: |
    .:53 {
      errors
      health {
        lameduck 5s
      }
      ready
      kubernetes cluster.local in-addr.arpa ip6.arpa {
        pods insecure
        fallthrough in-addr.arpa ip6.arpa
        ttl 30
      }
      prometheus :9153
      forward . /etc/resolv.conf
      cache 30
      loop
      reload
      loadbalance
    }
```

The Corefile configuration includes the following [plugins](#) of CoreDNS:

- [errors](#): Errors are logged to `stdout`.
- [health](#): Health of CoreDNS is reported to `http://localhost:8080/health`. In this extended syntax `lameduck` will make the process unhealthy then wait for 5 seconds before the process is shut down.
- [ready](#): An HTTP endpoint on port 8181 will return 200 OK, when all plugins that are able to signal readiness have done so.
- [kubernetes](#): CoreDNS will reply to DNS queries based on IP of the services and pods of Kubernetes. You can find [more details](#) about that

*plugin on the CoreDNS website. ttl allows you to set a custom TTL for responses. The default is 5 seconds. The minimum TTL allowed is 0 seconds, and the maximum is capped at 3600 seconds. Setting TTL to 0 will prevent records from being cached.*

*The pods insecure option is provided for backward compatibility with kube-dns. You can use the pods verified option, which returns an A record only if there exists a pod in same namespace with matching IP. The pods disabled option can be used if you don't use pod records.*

- [prometheus](#): Metrics of CoreDNS are available at `http://localhost:9153/metrics` in [Prometheus](#) format (also known as OpenMetrics).
- [forward](#): Any queries that are not within the cluster domain of Kubernetes will be forwarded to predefined resolvers (`/etc/resolv.conf`).
- [cache](#): This enables a frontend cache.
- [loop](#): Detects simple forwarding loops and halts the CoreDNS process if a loop is found.
- [reload](#): Allows automatic reload of a changed Corefile. After you edit the ConfigMap configuration, allow two minutes for your changes to take effect.
- [loadbalance](#): This is a round-robin DNS loadbalancer that randomizes the order of A, AAAA, and MX records in the answer.

You can modify the default CoreDNS behavior by modifying the ConfigMap.

## **Configuration of Stub-domain and upstream nameserver using CoreDNS**

CoreDNS has the ability to configure stubdomains and upstream nameservers using the [forward plugin](#).

### **Example**

If a cluster operator has a [Consul](#) domain server located at 10.150.0.1, and all Consul names have the suffix `.consul.local`. To configure it in CoreDNS, the cluster administrator creates the following stanza in the CoreDNS ConfigMap.

```
consul.local:53 {
    errors
    cache 30
    forward . 10.150.0.1
}
```

To explicitly force all non-cluster DNS lookups to go through a specific nameserver at 172.16.0.1, point the `forward` to the nameserver instead of `/etc/resolv.conf`

```
forward . 172.16.0.1
```

The final ConfigMap along with the default Corefile configuration looks like:

```

apiVersion: v1
kind: ConfigMap
metadata:
  name: coredns
  namespace: kube-system
data:
  Corefile: |
    .:53 {
      errors
      health
      kubernetes cluster.local in-addr.arpa ip6.arpa {
        pods insecure
        fallthrough in-addr.arpa ip6.arpa
      }
      prometheus :9153
      forward . 172.16.0.1
      cache 30
      loop
      reload
      loadbalance
    }
    consul.local:53 {
      errors
      cache 30
      forward . 10.150.0.1
    }
  }

```

The `kubeadm` tool supports automatic translation from the `kube-dns` `ConfigMap` to the equivalent `CoreDNS ConfigMap`.

**Note:** While `kube-dns` accepts an FQDN for `stubdomain` and `nameserver` (eg: `ns.foo.com`), `CoreDNS` does not support this feature. During translation, all FQDN nameservers will be omitted from the `CoreDNS` config.

## CoreDNS configuration equivalent to kube-dns

`CoreDNS` supports the features of `kube-dns` and more. A `ConfigMap` created for `kube-dns` to support `StubDomains` and `upstreamNameservers` translates to the `forward` plugin in `CoreDNS`. Similarly, the `Federations` plugin in `kube-dns` translates to the `federation` plugin in `CoreDNS`.

### Example

This example `ConfigMap` for `kube-dns` specifies `federations`, `stubdomains` and `upstreamnameservers`:

```

apiVersion: v1
data:
  federations: |

```

```

  {"foo" : "foo.feddomain.com"}
stubDomains: |
  {"abc.com" : ["1.2.3.4"], "my.cluster.local" : ["2.3.4.5"]}
upstreamNameservers: |
  ["8.8.8.8", "8.8.4.4"]
kind: ConfigMap

```

The equivalent configuration in CoreDNS creates a Corefile:

- For federations:

```
federation cluster.local {
    foo foo.feddomain.com
}
```

- For stubDomains:

```
abc.com:53 {
    errors
    cache 30
    forward . 1.2.3.4
}
my.cluster.local:53 {
    errors
    cache 30
    forward . 2.3.4.5
}
```

The complete Corefile with the default plugins:

```
.:53 {
    errors
    health
    kubernetes cluster.local in-addr.arpa ip6.arpa {
        pods insecure
        fallthrough in-addr.arpa ip6.arpa
    }
    federation cluster.local {
        foo foo.feddomain.com
    }
    prometheus :9153
    forward . 8.8.8.8 8.8.4.4
    cache 30
}
abc.com:53 {
    errors
    cache 30
    forward . 1.2.3.4
}
my.cluster.local:53 {
    errors
    cache 30
}
```

```
        forward . 2.3.4.5  
    }
```

## Migration to CoreDNS

To migrate from `kube-dns` to `CoreDNS`, a detailed [blog article](#) is available to help users adapt `CoreDNS` in place of `kube-dns`.

You can also migrate using the official `CoreDNS` [deploy script](#).

## What's next

- Read [Debugging DNS Resolution](#)

## 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](#).

Last modified August 07, 2020 at 8:40 PM PST: [Tune links in tasks section \(2/2\) \(92ae1a9cf\)](#)

[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Introduction](#)
- [CoreDNS](#)
  - [CoreDNS ConfigMap options](#)
  - [Configuration of Stub-domain and upstream nameserver using CoreDNS](#)
- [CoreDNS configuration equivalent to kube-dns](#)
  - [Example](#)
- [Migration to CoreDNS](#)
- [What's next](#)

## Debugging DNS Resolution

This page provides hints on diagnosing DNS problems.

## Before you begin

You need to have a Kubernetes cluster, and the `kubectl` command-line tool must be configured to communicate with your cluster. If you do not already

have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:

- [Katacoda](#)
- [Play with Kubernetes](#)

Your cluster must be configured to use the CoreDNS [addon](#) or its precursor, `kube-dns`.

Your Kubernetes server must be at or later than version v1.6. To check the version, enter `kubectl version`.

## Create a simple Pod to use as a test environment

[admin/dns/dnsutils.yaml](#)



```
apiVersion: v1
kind: Pod
metadata:
  name: dnsutils
  namespace: default
spec:
  containers:
  - name: dnsutils
    image: gcr.io/kubernetes-e2e-test-images/dnsutils:1.3
    command:
    - sleep
    - "3600"
    imagePullPolicy: IfNotPresent
  restartPolicy: Always
```

Use that manifest to create a Pod:

```
kubectl apply -f https://k8s.io/examples/admin/dns/dnsutils.yaml
```

```
pod/dnsutils created
```

and verify its status:

```
kubectl get pods dnsutils
```

NAME	READY	STATUS	RESTARTS	AGE
dnsutils	1/1	Running	0	<some-time>

Once that Pod is running, you can exec `nslookup` in that environment. If you see something like the following, DNS is working correctly.

```
kubectl exec -i -t dnsutils -- nslookup kubernetes.default
```

```
Server: 10.0.0.10
Address 1: 10.0.0.10

Name: kubernetes.default
Address 1: 10.0.0.1
```

If the `nslookup` command fails, check the following:

## **Check the local DNS configuration first**

Take a look inside the `resolv.conf` file. (See [Inheriting DNS from the node](#) and [Known issues](#) below for more information)

```
kubectl exec -ti dnsutils -- cat /etc/resolv.conf
```

Verify that the search path and name server are set up like the following (note that search path may vary for different cloud providers):

```
search default.svc.cluster.local svc.cluster.local cluster.local
google.internal c.gce_project_id.internal
nameserver 10.0.0.10
options ndots:5
```

Errors such as the following indicate a problem with the CoreDNS (or `kube-dns`) add-on or with associated Services:

```
kubectl exec -i -t dnsutils -- nslookup kubernetes.default
```

```
Server: 10.0.0.10
Address 1: 10.0.0.10

nslookup: can't resolve 'kubernetes.default'
```

or

```
kubectl exec -i -t dnsutils -- nslookup kubernetes.default
```

```
Server: 10.0.0.10
Address 1: 10.0.0.10 kube-dns.kube-system.svc.cluster.local
nslookup: can't resolve 'kubernetes.default'
```

## **Check if the DNS pod is running**

Use the `kubectl get pods` command to verify that the DNS pod is running.

```
kubectl get pods --namespace=kube-system -l k8s-app=kube-dns
```

NAME	READY	STATUS	RESTARTS	AGE
...				
coredns-7b96bf9f76-5hsxb	1/1	Running	0	1h
coredns-7b96bf9f76-mvmmmt	1/1	Running	0	1h
...				

**Note:** The value for label `k8s-app` is `kube-dns` for both CoreDNS and `kube-dns` deployments.

If you see that no CoreDNS Pod is running or that the Pod has failed/completed, the DNS add-on may not be deployed by default in your current environment and you will have to deploy it manually.

## **Check for errors in the DNS pod**

Use the `kubectl logs` command to see logs for the DNS containers.

For CoreDNS:

```
kubectl logs --namespace=kube-system -l k8s-app=kube-dns
```

Here is an example of a healthy CoreDNS log:

```
.:53
2018/08/15 14:37:17 [INFO] CoreDNS-1.2.2
2018/08/15 14:37:17 [INFO] linux/amd64, go1.10.3, 2e322f6
CoreDNS-1.2.2
linux/amd64, go1.10.3, 2e322f6
2018/08/15 14:37:17 [INFO] plugin/reload: Running configuration
MD5 = 24e6c59e83ce706f07bcc82c31b1ealc
```

*See if there are any suspicious or unexpected messages in the logs.*

## **Is DNS service up?**

*Verify that the DNS service is up by using the `kubectl get service` command.*

```
kubectl get svc --namespace=kube-system
```

NAME PORT(S)	TYPE	CLUSTER-IP	EXTERNAL-IP
		AGE	
...			
<i>kube-dns</i>	<i>ClusterIP</i>	<i>10.0.0.10</i>	<i>&lt;none&gt;</i>
TCP	1h		53/UDP, 53/
...			

**Note:** The service name is *kube-dns* for both CoreDNS and *kube-dns* deployments.

*If you have created the Service or in the case it should be created by default but it does not appear, see [debugging Services](#) for more information.*

## **Are DNS endpoints exposed?**

*You can verify that DNS endpoints are exposed by using the `kubectl get endpoints` command.*

```
kubectl get endpoints kube-dns --namespace=kube-system
```

NAME	ENDPOINTS	AGE
<i>kube-dns</i>	<i>10.180.3.17:53, 10.180.3.17:53</i>	<i>1h</i>

*If you do not see the endpoints, see the endpoints section in the [debugging Services](#) documentation.*

*For additional Kubernetes DNS examples, see the [cluster-dns examples](#) in the Kubernetes GitHub repository.*

## **Are DNS queries being received/processed?**

You can verify if queries are being received by CoreDNS by adding the `log` plugin to the CoreDNS configuration (aka `Corefile`). The CoreDNS `Corefile` is held in a [ConfigMap](#) named `coredns`. To edit it, use the command:

```
kubectl -n kube-system edit configmap coredns
```

Then add `log` in the `Corefile` section per the example below:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: coredns
  namespace: kube-system
data:
  Corefile: |
    .:53 {
      log
      errors
      health
      kubernetes cluster.local in-addr.arpa ip6.arpa {
        pods insecure
        upstream
        fallthrough in-addr.arpa ip6.arpa
      }
      prometheus :9153
      forward . /etc/resolv.conf
      cache 30
      loop
      reload
      loadbalance
    }
```

After saving the changes, it may take up to minute or two for Kubernetes to propagate these changes to the CoreDNS pods.

Next, make some queries and view the logs per the sections above in this document. If CoreDNS pods are receiving the queries, you should see them in the logs.

Here is an example of a query in the log:

```
.:53
2018/08/15 14:37:15 [INFO] CoreDNS-1.2.0
2018/08/15 14:37:15 [INFO] linux/amd64, go1.10.3, 2e322f6
CoreDNS-1.2.0
linux/amd64, go1.10.3, 2e322f6
2018/09/07 15:29:04 [INFO] plugin/reload: Running configuration
```

```
MD5 = 162475cdf272d8aa601e6fe67a6ad42f
2018/09/07 15:29:04 [INFO] Reloading complete
172.17.0.18:41675 - [07/Sep/2018:15:29:11 +0000] 59925 "A IN
kubernetes.default.svc.cluster.local. udp 54 false 512" NOERROR
qr,aa,rd,ra 106 0.000066649s
```

## Known issues

*Some Linux distributions (e.g. Ubuntu) use a local DNS resolver by default (`systemd-resolved`). `Systemd-resolved` moves and replaces `/etc/resolv.conf` with a stub file that can cause a fatal forwarding loop when resolving names in upstream servers. This can be fixed manually by using `kubelet`'s `--resolv-conf` flag to point to the correct `resolv.conf` (With `systemd-resolved`, this is `/run/systemd/resolve/resolv.conf`). `kubeadm` automatically detects `systemd-resolved`, and adjusts the `kubelet` flags accordingly.*

*Kubernetes installs do not configure the nodes' `resolv.conf` files to use the cluster DNS by default, because that process is inherently distribution-specific. This should probably be implemented eventually.*

*Linux's `libc` (a.k.a. `glibc`) has a limit for the DNS nameserver records to 3 by default. What's more, for the `glibc` versions which are older than `glibc-2.17-222` ([the new versions update see this issue](#)), the allowed number of DNS search records has been limited to 6 ([see this bug from 2005](#)).*

*Kubernetes needs to consume 1 nameserver record and 3 search records. This means that if a local installation already uses 3 nameservers or uses more than 3 searches while your `glibc` version is in the affected list, some of those settings will be lost. To work around the DNS nameserver records limit, the node can run `dnsmasq`, which will provide more nameserver entries. You can also use `kubelet`'s `--resolv-conf` flag. To fix the DNS search records limit, consider upgrading your linux distribution or upgrading to an unaffected version of `glibc`.*

*If you are using Alpine version 3.3 or earlier as your base image, DNS may not work properly due to a known issue with Alpine. Kubernetes [issue 30215](#) details more information on this.*

## What's next

- See [Autoscaling the DNS Service in a Cluster](#).
- Read [DNS for Services and Pods](#)

# 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](#).

Last modified October 11, 2020 at 4:27 PM PST: [doc: update corefile \(087513d4c\)](#)

[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
  - [Create a simple Pod to use as a test environment](#)
  - [Check the local DNS configuration first](#)
  - [Check if the DNS pod is running](#)
  - [Check for errors in the DNS pod](#)
  - [Is DNS service up?](#)
  - [Are DNS endpoints exposed?](#)
  - [Are DNS queries being received/processed?](#)
- [Known issues](#)
- [What's next](#)

# Declare Network Policy

This document helps you get started using the Kubernetes [NetworkPolicy API](#) to declare network policies that govern how pods communicate with each other.

**Caution:** This section links to third party projects that provide functionality required by Kubernetes. The Kubernetes project authors aren't responsible for these projects. This page follows [CNCF website guidelines](#) by listing projects alphabetically. To add a project to this list, read the [content guide](#) before submitting a change.

## Before you begin

You need to have a Kubernetes cluster, and the `kubectl` command-line tool must be configured to communicate with your cluster. If you do not already have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:

- [Katacoda](#)
- [Play with Kubernetes](#)

Your Kubernetes server must be at or later than version v1.8. To check the version, enter `kubectl version`.

Make sure you've configured a network provider with network policy support. There are a number of network providers that support `NetworkPolicy`, including:

- [Calico](#)
- [Cilium](#)
- [Kube-router](#)
- [Romana](#)
- [Weave Net](#)

## Create an `nginx` deployment and expose it via a service

To see how Kubernetes network policy works, start off by creating an `nginx` Deployment.

```
kubectl create deployment nginx --image=nginx
```

```
deployment.apps/nginx created
```

Expose the Deployment through a Service called `nginx`.

```
kubectl expose deployment nginx --port=80
```

```
service/nginx exposed
```

The above commands create a Deployment with an `nginx` Pod and expose the Deployment through a Service named `nginx`. The `nginx` Pod and Deployment are found in the default namespace.

```
kubectl get svc,pod
```

NAME	CLUSTER-IP	EXTERNAL-IP
PORT(S)	AGE	
service/kubernetes	10.100.0.1	<none>
TCP 46m		443/
service/nginx	10.100.0.16	<none>
TCP 33s		80/

NAME	READY	STATUS
------	-------	--------

RESTARTS	AGE		
pod/nginx-701339712-e0qfq	1/1		Running
0	35s		

## **Test the service by accessing it from another Pod**

You should be able to access the new `nginx` service from other Pods. To access the `nginx` Service from another Pod in the `default` namespace, start a `busybox` container:

```
kubectl run busybox --rm -ti --image=busybox -- /bin/sh
```

In your shell, run the following command:

```
wget --spider --timeout=1 nginx
```

```
Connecting to nginx (10.100.0.16:80)
remote file exists
```

## **Limit access to the `nginx` service**

To limit the access to the `nginx` service so that only Pods with the label `access: true` can query it, create a `NetworkPolicy` object as follows:

[service/networking/nginx-policy.yaml](#)  


```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: access-nginx
spec:
  podSelector:
    matchLabels:
      app: nginx
  ingress:
  - from:
    - podSelector:
        matchLabels:
          access: "true"
```

The name of a `NetworkPolicy` object must be a valid [DNS subdomain name](#).

**Note:** *NetworkPolicy* includes a *podSelector* which selects the grouping of Pods to which the policy applies. You can see this policy selects Pods with the label *app=nginx*. The label was automatically added to the Pod in the *nginx* Deployment. An empty *podSelector* selects all pods in the namespace.

## **Assign the policy to the service**

Use *kubectl* to create a *NetworkPolicy* from the above *nginx-policy.yaml* file:

```
kubectl apply -f https://k8s.io/examples/service/networking/nginx-policy.yaml
```

```
networkpolicy.networking.k8s.io/access-nginx created
```

## **Test access to the service when access label is not defined**

When you attempt to access the *nginx* Service from a Pod without the correct labels, the request times out:

```
kubectl run busybox --rm -ti --image=busybox -- /bin/sh
```

In your shell, run the command:

```
wget --spider --timeout=1 nginx
```

```
Connecting to nginx (10.100.0.16:80)
wget: download timed out
```

## **Define access label and test again**

You can create a Pod with the correct labels to see that the request is allowed:

```
kubectl run busybox --rm -ti --labels="access=true" --image=busybox -- /bin/sh
```

In your shell, run the command:

```
wget --spider --timeout=1 nginx
```

```
Connecting to nginx (10.100.0.16:80)
remote file exists
```

## 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](#).

Last modified August 11, 2020 at 10:23 AM PST: [Add 3rd party content warning \(47dd26bf0\)](#)

[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Create an nginx deployment and expose it via a service](#)
- [Test the service by accessing it from another Pod](#)
- [Limit access to the nginx service](#)
- [Assign the policy to the service](#)
- [Test access to the service when access label is not defined](#)
- [Define access label and test again](#)

# Developing Cloud Controller Manager

**FEATURE STATE:** Kubernetes v1.11 [beta]

The `cloud-controller-manager` is a Kubernetes [control plane](#) component that embeds cloud-specific control logic. The cloud controller manager lets you link your cluster into your cloud provider's API, and separates out the components that interact with that cloud platform from components that just interact with your cluster.

By decoupling the interoperability logic between Kubernetes and the underlying cloud infrastructure, the `cloud-controller-manager` component enables cloud providers to release features at a different pace compared to the main Kubernetes project.

# **Background**

*Since cloud providers develop and release at a different pace compared to the Kubernetes project, abstracting the provider-specific code to the cloud-controller-manager binary allows cloud vendors to evolve independently from the core Kubernetes code.*

*The Kubernetes project provides skeleton cloud-controller-manager code with Go interfaces to allow you (or your cloud provider) to plug in your own implementations. This means that a cloud provider can implement a cloud-controller-manager by importing packages from Kubernetes core; each cloudprovider will register their own code by calling `cloudprovider.RegisterCloudProvider` to update a global variable of available cloud providers.*

# **Developing**

## **Out of tree**

*To build an out-of-tree cloud-controller-manager for your cloud:*

1. Create a go package with an implementation that satisfies [`cloudprovider.Interface`](#).
2. Use [`main.go` in cloud-controller-manager](#) from Kubernetes core as a template for your `main.go`. As mentioned above, the only difference should be the `cloud` package that will be imported.
3. Import your `cloud` package in `main.go`, ensure your package has an `init` block to run [`cloudprovider.RegisterCloudProvider`](#).

*Many cloud providers publish their controller manager code as open source. If you are creating a new cloud-controller-manager from scratch, you could take an existing out-of-tree cloud controller manager as your starting point.*

## **In tree**

*For in-tree cloud providers, you can run the in-tree cloud controller manager as a [`DaemonSet`](#) in your cluster. See [`Cloud Controller Manager Administration`](#) for more details.*

# **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](#).*

*Last modified May 30, 2020 at 3:10 PM PST: [add en pages \(ecc27bbbe\)](#)*

- [Background](#)
- [Developing](#)
  - [Out of tree](#)
  - [In tree](#)

# Enable Or Disable A Kubernetes API

This page shows how to enable or disable an API version from your cluster's [control plane](#).

Specific API versions can be turned on or off by passing `--runtime-config=api/<version>` as a command line argument to the API server. The values for this argument are a comma-separated list of API versions. Later values override earlier values.

The `runtime-config` command line argument also supports 2 special keys:

- `api/all`, representing all known APIs
- `api/legacy`, representing only legacy APIs. Legacy APIs are any APIs that have been explicitly [deprecated](#).

For example, to turning off all API versions except v1, pass `--runtime-config=api/all=false,api/v1=true` to the `kube-apiserver`.

## What's next

Read the [full documentation](#) for the `kube-apiserver` component.

## 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](#).

Last modified October 07, 2020 at 7:16 PM PST: [Revise cluster management task \(59dcd57cc\)](#)

[Edit this page](#) [Create child page](#) [Create an issue](#)

- [What's next](#)

# **Enabling EndpointSlices**

*This page provides an overview of enabling EndpointSlices in Kubernetes.*

## **Before you begin**

*You need to have a Kubernetes cluster, and the `kubectl` command-line tool must be configured to communicate with your cluster. If you do not already have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:*

- [Katacoda](#)
- [Play with Kubernetes](#)

*To check the version, enter `kubectl version`.*

## **Introduction**

*EndpointSlices provide a scalable and extensible alternative to Endpoints in Kubernetes. They build on top of the base of functionality provided by Endpoints and extend that in a scalable way. When Services have a large number (>100) of network endpoints, they will be split into multiple smaller EndpointSlice resources instead of a single large Endpoints resource.*

## **Enabling EndpointSlices**

**FEATURE STATE:** Kubernetes v1.17 [beta]

**Note:** The EndpointSlice resource was designed to address shortcomings in a earlier resource: Endpoints. Some Kubernetes components and third-party applications continue to use and rely on Endpoints. Whilst that remains the case, EndpointSlices should be seen as an addition to Endpoints in a cluster, not as an outright replacement.

*EndpointSlice functionality in Kubernetes is made up of several different components, most are enabled by default:*

- *The EndpointSlice API: EndpointSlices are part of the `discovery.k8s.io/v1beta1` API. This is beta and enabled by default since Kubernetes 1.17. All components listed below are dependent on this API being enabled.*
- *The EndpointSlice Controller: This [controller](#) maintains EndpointSlices for Services and the Pods they reference. This is controlled by the Endp*

*ointSlice feature gate. It has been enabled by default since Kubernetes 1.18.*

- *The EndpointSliceMirroring Controller: This [controller](#) mirrors custom Endpoints to EndpointSlices. This is controlled by the EndpointSlice feature gate. It has been enabled by default since Kubernetes 1.19.*
- *Kube-Proxy: When [kube-proxy](#) is configured to use EndpointSlices, it can support higher numbers of Service endpoints. This is controlled by the EndpointSliceProxying feature gate on Linux and WindowsEndpointSliceProxying on Windows. It has been enabled by default on Linux since Kubernetes 1.19. It is not enabled by default for Windows nodes. To configure kube-proxy to use EndpointSlices on Windows, you can enable the WindowsEndpointSliceProxying [feature gate](#) on kube-proxy.*

## API fields

*Some fields in the EndpointSlice API are feature-gated.*

- *The EndpointSliceNodeName feature gate controls access to the nodeName field. This is an alpha feature that is disabled by default.*
- *The EndpointSliceTerminating feature gate controls access to the serving and terminating condition fields. This is an alpha feature that is disabled by default.*

## Using EndpointSlices

*With EndpointSlices fully enabled in your cluster, you should see corresponding EndpointSlice resources for each Endpoints resource. In addition to supporting existing Endpoints functionality, EndpointSlices will allow for greater scalability and extensibility of network endpoints in your cluster.*

## What's next

- Read about [EndpointSlices](#)
- Read [Connecting Applications with Services](#)

## 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](#).

Last modified November 19, 2020 at 6:26 PM PST: [Updating EndpointSlice docs for Kubernetes 1.20 \(c927e9a57\)](#)

[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Introduction](#)
- [Enabling EndpointSlices](#)
- [API fields](#)
- [Using EndpointSlices](#)
- [What's next](#)

# Enabling Service Topology

This page provides an overview of enabling Service Topology in Kubernetes.

## Before you begin

You need to have a Kubernetes cluster, and the `kubectl` command-line tool must be configured to communicate with your cluster. If you do not already have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:

- [Katacoda](#)
- [Play with Kubernetes](#)

To check the version, enter `kubectl version`.

## Introduction

Service Topology enables a service to route traffic based upon the Node topology of the cluster. For example, a service can specify that traffic be preferentially routed to endpoints that are on the same Node as the client, or in the same availability zone.

## Prerequisites

The following prerequisites are needed in order to enable topology aware service routing:

- Kubernetes 1.17 or later
- [Kube-proxy](#) running in iptables mode or IPVS mode

- Enable [Endpoint Slices](#)

## Enable Service Topology

**FEATURE STATE:** Kubernetes v1.17 [alpha]

To enable service topology, enable the `ServiceTopology` and `EndpointSlice` feature gate for all Kubernetes components:

```
--feature-gates="ServiceTopology=true,EndpointSlice=true"
```

## What's next

- Read about the [Service Topology](#) concept
- Read about [Endpoint Slices](#)
- Read [Connecting Applications with Services](#)

## 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](#).

Last modified May 30, 2020 at 3:10 PM PST: [add en pages \(ecc27bbbe\)](#)  
[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Introduction](#)
- [Prerequisites](#)
- [Enable Service Topology](#)
- [What's next](#)

## Encrypting Secret Data at Rest

This page shows how to enable and configure encryption of secret data at rest.

### Before you begin

-

You need to have a Kubernetes cluster, and the `kubectl` command-line tool must be configured to communicate with your cluster. If you do not already have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:

- [Katacoda](#)
- [Play with Kubernetes](#)

Your Kubernetes server must be at or later than version 1.13. To check the version, enter `kubectl version`.

- etcd v3.0 or later is required

## **Configuration and determining whether encryption at rest is already enabled**

The `kube-apiserver` process accepts an argument `--encryption-provider-config` that controls how API data is encrypted in etcd. An example configuration is provided below.

## **Understanding the encryption at rest configuration.**

```
apiVersion: apiserver.config.k8s.io/v1
kind: EncryptionConfiguration
resources:
  - resources:
    - secrets
  providers:
    - identity: {}
    - aesgcm:
        keys:
          - name: key1
            secret: c2VjcmV0IGlzIHNlY3VyZQ==
          - name: key2
            secret: dGhpccyBpcyBwYXNzd29yZA==
    - aescbc:
        keys:
          - name: key1
            secret: c2VjcmV0IGlzIHNlY3VyZQ==
          - name: key2
            secret: dGhpccyBpcyBwYXNzd29yZA==
    - secretbox:
        keys:
          - name: key1
            secret: YWJjZGVmZ2hpamtsbW5vcHFyc3R1dnd4eXoxMjM0NTY=
```

Each `resources` array item is a separate config and contains a complete configuration. The `resources.resources` field is an array of Kubernetes resource names (resource or resource.group) that should be encrypted. The `providers` array is an ordered list of the possible encryption providers.

*Only one provider type may be specified per entry (`identity` or `aescbc` may be provided, but not both in the same item).*

*The first provider in the list is used to encrypt resources going into storage. When reading resources from storage each provider that matches the stored data attempts to decrypt the data in order. If no provider can read the stored data due to a mismatch in format or secret key, an error is returned which prevents clients from accessing that resource.*

**Caution: IMPORTANT:** *If any resource is not readable via the encryption config (because keys were changed), the only recourse is to delete that key from the underlying etcd directly. Calls that attempt to read that resource will fail until it is deleted or a valid decryption key is provided.*

## **Providers:**

Name	Encryption	Strength	Speed	Key Length	Other Considerations
identity	None	N/A	N/A	N/A	Resources written as-is without encryption. When set as the first provider, the resource will be decrypted as new values are written.
aescbc	AES-CBC with PKCS#7 padding	Strongest	Fast	32-byte	The recommended choice for encryption at rest but may be slightly slower than <code>secretbox</code> .
secretbox	XSalsa20 and Poly1305	Strong	Faster	32-byte	A newer standard and may not be considered acceptable in environments that require high levels of review.
aesgcm	AES-GCM with random nonce	Must be rotated every 200k writes	Fastest	16, 24, or 32-byte	Is not recommended for use except when an automated key rotation scheme is implemented.

Name	Encryption	Strength	Speed	Key Length	Other Considerations
kms	Uses envelope encryption scheme: Data is encrypted by data encryption keys (DEKs) using AES-CBC with PKCS#7 padding, DEKs are encrypted by key encryption keys (KEKs) according to configuration in Key Management Service (KMS)	Strongest	Fast	32-bytes	The recommended choice for using a third party tool for key management. Simplifies key rotation, with a new DEK generated for each encryption, and KEK rotation controlled by the user. <a href="#">Configure the KMS provider</a>

*Each provider supports multiple keys - the keys are tried in order for decryption, and if the provider is the first provider, the first key is used for encryption.*

***Storing the raw encryption key in the EncryptionConfig only moderately improves your security posture, compared to no encryption. Please use kms provider for additional security.*** By default, the identity provider is used to protect secrets in etcd, which provides no encryption. EncryptionConfiguration was introduced to encrypt secrets locally, with a locally managed key.

*Encrypting secrets with a locally managed key protects against an etcd compromise, but it fails to protect against a host compromise. Since the encryption keys are stored on the host in the EncryptionConfig YAML file, a skilled attacker can access that file and extract the encryption keys.*

*Envelope encryption creates dependence on a separate key, not stored in Kubernetes. In this case, an attacker would need to compromise etcd, the kubeapi-server, and the third-party KMS provider to retrieve the plaintext values, providing a higher level of security than locally-stored encryption keys.*

## Encrypting your data

Create a new encryption config file:

```
apiVersion: apiserver.config.k8s.io/v1
kind: EncryptionConfiguration
resources:
  - resources:
    - secrets
  providers:
```

```
- aescbc:
  keys:
    - name: key1
      secret: <BASE 64 ENCODED SECRET>
    - identity: {}
```

To create a new secret perform the following steps:

1. Generate a 32 byte random key and base64 encode it. If you're on Linux or macOS, run the following command:

```
head -c 32 /dev/urandom | base64
```

2. Place that value in the secret field.
3. Set the `--encryption-provider-config` flag on the `kube-apiserver` to point to the location of the config file.
4. Restart your API server.

**Caution:** Your config file contains keys that can decrypt content in etcd, so you must properly restrict permissions on your masters so only the user who runs the `kube-apiserver` can read it.

## Verifying that data is encrypted

Data is encrypted when written to etcd. After restarting your `kube-apiserver`, any newly created or updated secret should be encrypted when stored. To check, you can use the `etcdctl` command line program to retrieve the contents of your secret.

1. Create a new secret called `secret1` in the `default` namespace:

```
kubectl create secret generic secret1 -n default --from-literal=mykey=mydata
```

2. Using the `etcdctl` commandline, read that secret out of etcd:

```
ETCDCTL_API=3 etcdctl get /registry/secrets/default/secret1 [...] | hexdump -C
```

where [...] must be the additional arguments for connecting to the etcd server.

3. Verify the stored secret is prefixed with `k8s:enc:aescbc:v1:` which indicates the `aescbc` provider has encrypted the resulting data.
4. Verify the secret is correctly decrypted when retrieved via the API:

```
kubectl describe secret secret1 -n default
```

should match `mykey: bXlkYXRh`, `mydata` is encoded, check [decoding a secret](#) to completely decode the secret.

## **Ensure all secrets are encrypted**

*Since secrets are encrypted on write, performing an update on a secret will encrypt that content.*

```
kubectl get secrets --all-namespaces -o json | kubectl replace -f -
```

*The command above reads all secrets and then updates them to apply server side encryption.*

**Note:** If an error occurs due to a conflicting write, retry the command. For larger clusters, you may wish to subdivide the secrets by namespace or script an update.

## **Rotating a decryption key**

*Changing the secret without incurring downtime requires a multi step operation, especially in the presence of a highly available deployment where multiple kube-apiserver processes are running.*

1. Generate a new key and add it as the second key entry for the current provider on all servers
2. Restart all kube-apiserver processes to ensure each server can decrypt using the new key
3. Make the new key the first entry in the keys array so that it is used for encryption in the config
4. Restart all kube-apiserver processes to ensure each server now encrypts using the new key
5. Run `kubectl get secrets --all-namespaces -o json | kubectl replace -f -` to encrypt all existing secrets with the new key
6. Remove the old decryption key from the config after you back up etcd with the new key in use and update all secrets

*With a single kube-apiserver, step 2 may be skipped.*

## **Decrypting all data**

*To disable encryption at rest place the identity provider as the first entry in the config:*

```
apiVersion: apiserver.config.k8s.io/v1
kind: EncryptionConfiguration
resources:
  - resources:
    - secrets
  providers:
    - identity: {}
    - aescbc:
      keys:
```

```
- name: key1  
secret: <BASE 64 ENCODED SECRET>
```

and restart all `kube-apiserver` processes. Then run:

```
kubectl get secrets --all-namespaces -o json | kubectl replace -f -
```

to force all secrets to be decrypted.

## 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](#).

Last modified May 30, 2020 at 3:10 PM PST: [add en pages \(ecc27bbbe\)](#)  
[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Configuration and determining whether encryption at rest is already enabled](#)
- [Understanding the encryption at rest configuration.](#)
  - [Providers:](#)

## Guaranteed Scheduling For Critical Add-On Pods

In addition to Kubernetes core components like api-server, scheduler, controller-manager running on a master machine there are a number of add-ons which, for various reasons, must run on a regular cluster node (rather than the Kubernetes master). Some of these add-ons are critical to a fully functional cluster, such as metrics-server, DNS, and UI. A cluster may stop working properly if a critical add-on is evicted (either manually or as a side effect of another operation like upgrade) and becomes pending (for example when the cluster is highly utilized and either there are other pending pods that schedule into the space vacated by the evicted critical add-on pod or the amount of resources available on the node changed for some other reason).

Note that marking a pod as critical is not meant to prevent evictions entirely; it only prevents the pod from becoming permanently unavailable. For static pods, this means it can't be evicted, but for non-static pods, it just means they will always be rescheduled.

## **Marking pod as critical**

To mark a Pod as critical, set priorityClassName for that Pod to `system-cluster-critical` or `system-node-critical`. `system-node-critical` is the highest available priority, even higher than `system-cluster-critical`.

## **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](#).

Last modified May 30, 2020 at 3:10 PM PST: [add en pages \(ecc27bbbe\)](#)  
[Edit this page](#) [Create child page](#) [Create an issue](#)

- ◦ [Marking pod as critical](#)

# **IP Masquerade Agent User Guide**

This page shows how to configure and enable the ip-masq-agent.

## **Before you begin**

You need to have a Kubernetes cluster, and the `kubectl` command-line tool must be configured to communicate with your cluster. If you do not already have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:

- [Katacoda](#)
- [Play with Kubernetes](#)

To check the version, enter `kubectl version`.

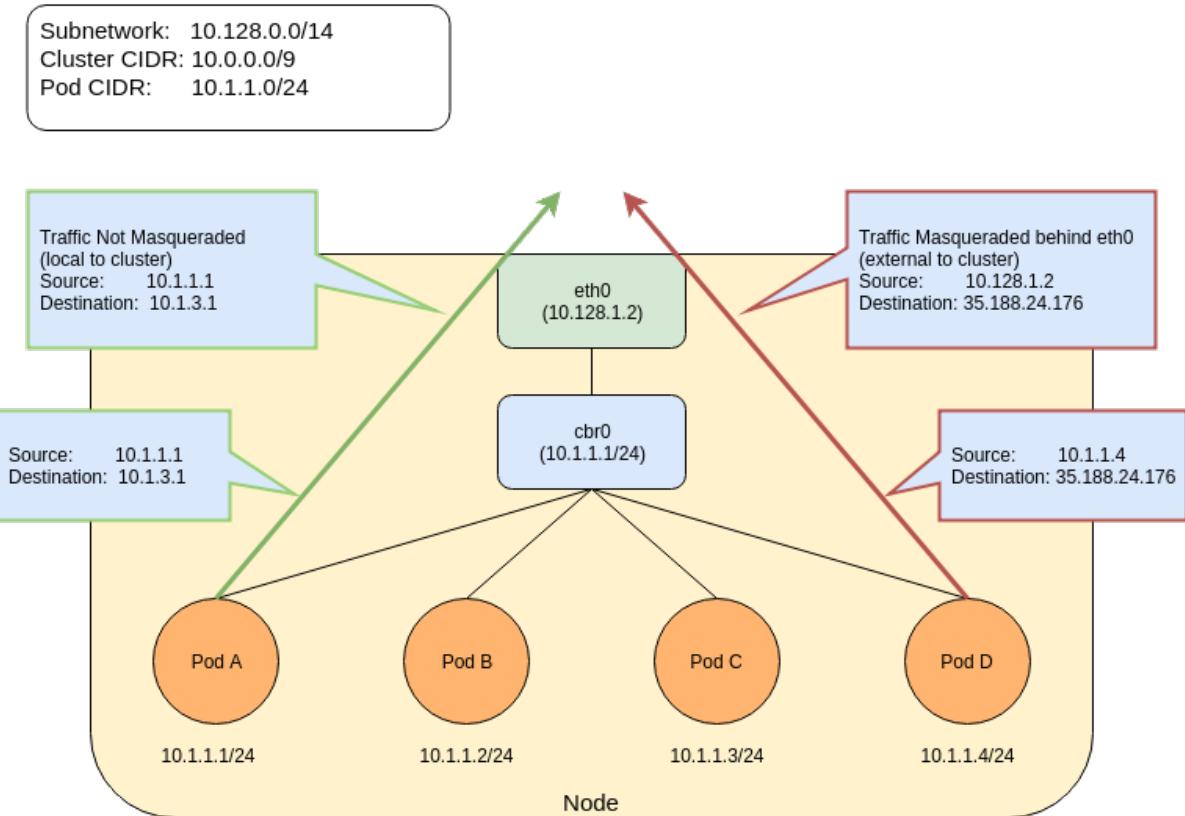
# **IP Masquerade Agent User Guide**

The ip-masq-agent configures iptables rules to hide a pod's IP address behind the cluster node's IP address. This is typically done when sending traffic to destinations outside the cluster's pod [CIDR](#) range.

## **Key Terms**

- **NAT (Network Address Translation)** Is a method of remapping one IP address to another by modifying either the source and/or destination address information in the IP header. Typically performed by a device doing IP routing.
- **Masquerading** A form of NAT that is typically used to perform a many to one address translation, where multiple source IP addresses are masked behind a single address, which is typically the device doing the IP routing. In Kubernetes this is the Node's IP address.
- **CIDR (Classless Inter-Domain Routing)** Based on the variable-length subnet masking, allows specifying arbitrary-length prefixes. CIDR introduced a new method of representation for IP addresses, now commonly known as **CIDR notation**, in which an address or routing prefix is written with a suffix indicating the number of bits of the prefix, such as 192.168.2.0/24.
- **Link Local** A link-local address is a network address that is valid only for communications within the network segment or the broadcast domain that the host is connected to. Link-local addresses for IPv4 are defined in the address block 169.254.0.0/16 in CIDR notation.

The ip-masq-agent configures iptables rules to handle masquerading node/pod IP addresses when sending traffic to destinations outside the cluster node's IP and the Cluster IP range. This essentially hides pod IP addresses behind the cluster node's IP address. In some environments, traffic to "external" addresses must come from a known machine address. For example, in Google Cloud, any traffic to the internet must come from a VM's IP. When containers are used, as in Google Kubernetes Engine, the Pod IP will be rejected for egress. To avoid this, we must hide the Pod IP behind the VM's own IP address - generally known as "masquerade". By default, the agent is configured to treat the three private IP ranges specified by [RFC 1918](#) as non-masquerade **CIDR**. These ranges are 10.0.0.0/8, 172.16.0.0/12, and 192.168.0.0/16. The agent will also treat link-local (169.254.0.0/16) as a non-masquerade CIDR by default. The agent is configured to reload its configuration from the location /etc/config/ip-masq-agent every 60 seconds, which is also configurable.



The agent configuration file must be written in YAML or JSON syntax, and may contain three optional keys:

- **nonMasqueradeCIDRs**: A list of strings in [CIDR](#) notation that specify the non-masquerade ranges.
- **masqLinkLocal**: A Boolean (true / false) which indicates whether to masquerade traffic to the link local prefix 169.254.0.0/16. False by default.
- **resyncInterval**: A time interval at which the agent attempts to reload config from disk. For example: '30s', where 's' means seconds, 'ms' means milliseconds, etc...

Traffic to 10.0.0.0/8, 172.16.0.0/12 and 192.168.0.0/16) ranges will NOT be masqueraded. Any other traffic (assumed to be internet) will be masqueraded. An example of a local destination from a pod could be its Node's IP address as well as another node's address or one of the IP addresses in Cluster's IP range. Any other traffic will be masqueraded by default. The below entries show the default set of rules that are applied by the ip-masq-agent:

```
iptables -t nat -L IP-MASQ-AGENT
RETURN      all  --  anywhere           169.254.0.0/16          /*
ip-masq-agent: cluster-local traffic should not be subject to
MASQUERADE */ ADDRTYPE match dst-type !LOCAL
```

```

RETURN    all  --  anywhere          10.0.0.0/8      /*
ip-masq-agent: cluster-local traffic should not be subject to
MASQUERADE */ ADDRTYPE match dst-type !LOCAL
RETURN    all  --  anywhere          172.16.0.0/12     /*
ip-masq-agent: cluster-local traffic should not be subject to
MASQUERADE */ ADDRTYPE match dst-type !LOCAL
RETURN    all  --  anywhere          192.168.0.0/16     /*
ip-masq-agent: cluster-local traffic should not be subject to
MASQUERADE */ ADDRTYPE match dst-type !LOCAL
MASQUERADE all  --  anywhere      anywhere          /
* ip-masq-agent: outbound traffic should be subject to
MASQUERADE (this match must come after cluster-local CIDR
matches) */ ADDRTYPE match dst-type !LOCAL

```

*By default, in GCE/Google Kubernetes Engine starting with Kubernetes version 1.7.0, if network policy is enabled or you are using a cluster CIDR not in the 10.0.0.0/8 range, the ip-masq-agent will run in your cluster. If you are running in another environment, you can add the ip-masq-agent [DaemonSet](#) to your cluster:*

## Create an ip-masq-agent

*To create an ip-masq-agent, run the following kubectl command:*

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes-
sigs/ip-masq-agent/master/ip-masq-agent.yaml
```

*You must also apply the appropriate node label to any nodes in your cluster that you want the agent to run on.*

```
kubectl label nodes my-node beta.kubernetes.io/masq-agent-ds-
ready=true
```

*More information can be found in the ip-masq-agent documentation [here](#)*

*In most cases, the default set of rules should be sufficient; however, if this is not the case for your cluster, you can create and apply a [ConfigMap](#) to customize the IP ranges that are affected. For example, to allow only 10.0.0.0/8 to be considered by the ip-masq-agent, you can create the following [ConfigMap](#) in a file called "config".*

**Note:**

*It is important that the file is called config since, by default, that will be used as the key for lookup by the ip-masq-agent:*

```
nonMasqueradeCIDRs:  
  - 10.0.0.0/8  
resyncInterval: 60s
```

*Run the following command to add the config map to your cluster:*

```
kubectl create configmap ip-masq-agent --from-file=config --  
namespace=kube-system
```

*This will update a file located at /etc/config/ip-masq-agent which is periodically checked every resyncInterval and applied to the cluster node. After the resync interval has expired, you should see the iptables rules reflect your changes:*

```
iptables -t nat -L IP-MASQ-AGENT  
Chain IP-MASQ-AGENT (1 references)  
target      prot opt source          destination  
RETURN     all  --  anywhere       169.254.0.0/16      /*  
ip-masq-agent: cluster-local traffic should not be subject to  
MASQUERADE */ ADDRTYPE match dst-type !LOCAL  
RETURN     all  --  anywhere       10.0.0.0/8        /*  
ip-masq-agent: cluster-local  
MASQUERADE all  --  anywhere       anywhere          /  
* ip-masq-agent: outbound traffic should be subject to  
MASQUERADE (this match must come after cluster-local CIDR  
matches) */ ADDRTYPE match dst-type !LOCAL
```

*By default, the link local range (169.254.0.0/16) is also handled by the ip-masq agent, which sets up the appropriate iptables rules. To have the ip-masq-agent ignore link local, you can set masqLinkLocal to true in the config map.*

```
nonMasqueradeCIDRs:  
  - 10.0.0.0/8  
resyncInterval: 60s  
masqLinkLocal: true
```

## 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](#).

Last modified October 17, 2020 at 3:21 PM PST: [update kubernetes-incubator references \(a8b6551c2\)](#)

[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [IP Masquerade Agent User Guide](#)
  - [Key Terms](#)
- [Create an ip-masq-agent](#)

## **Limit Storage Consumption**

This example demonstrates an easy way to limit the amount of storage consumed in a namespace.

The following resources are used in the demonstration: [ResourceQuota](#), [LimitRange](#), and [PersistentVolumeClaim](#).

### **Before you begin**

- You need to have a Kubernetes cluster, and the `kubectl` command-line tool must be configured to communicate with your cluster. If you do not already have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:

- [Katacoda](#)
- [Play with Kubernetes](#)

To check the version, enter `kubectl version`.

### **Scenario: Limiting Storage Consumption**

The cluster-admin is operating a cluster on behalf of a user population and the admin wants to control how much storage a single namespace can consume in order to control cost.

The admin would like to limit:

1. The number of persistent volume claims in a namespace
2. The amount of storage each claim can request
3. The amount of cumulative storage the namespace can have

### **LimitRange to limit requests for storage**

Adding a `LimitRange` to a namespace enforces storage request sizes to a minimum and maximum. Storage is requested via `PersistentVolumeClaim`.

*The admission controller that enforces limit ranges will reject any PVC that is above or below the values set by the admin.*

*In this example, a PVC requesting 10Gi of storage would be rejected because it exceeds the 2Gi max.*

```
apiVersion: v1
kind: LimitRange
metadata:
  name: storagelimits
spec:
  limits:
    - type: PersistentVolumeClaim
      max:
        storage: 2Gi
      min:
        storage: 1Gi
```

*Minimum storage requests are used when the underlying storage provider requires certain minimums. For example, AWS EBS volumes have a 1Gi minimum requirement.*

## **StorageQuota to limit PVC count and cumulative storage capacity**

*Admins can limit the number of PVCs in a namespace as well as the cumulative capacity of those PVCs. New PVCs that exceed either maximum value will be rejected.*

*In this example, a 6th PVC in the namespace would be rejected because it exceeds the maximum count of 5. Alternatively, a 5Gi maximum quota when combined with the 2Gi max limit above, cannot have 3 PVCs where each has 2Gi. That would be 6Gi requested for a namespace capped at 5Gi.*

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: storagequota
spec:
  hard:
    persistentvolumeclaims: "5"
    requests.storage: "5Gi"
```

## **Summary**

*A limit range can put a ceiling on how much storage is requested while a resource quota can effectively cap the storage consumed by a namespace through claim counts and cumulative storage capacity. This allows a cluster-admin to plan their cluster's storage budget without risk of any one project going over their allotment.*

# 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](#).

Last modified July 29, 2020 at 9:40 PM PST: [Fix broken links to pages under /en/docs/tasks/administer-cluster/manage-resources/ \(36d9239fb\)](#)  
[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Scenario: Limiting Storage Consumption](#)
- [LimitRange to limit requests for storage](#)
- [StorageQuota to limit PVC count and cumulative storage capacity](#)
- [Summary](#)

# Namespaces Walkthrough

Kubernetes [namespaces](#) help different projects, teams, or customers to share a Kubernetes cluster.

It does this by providing the following:

1. A scope for [Names](#).
2. A mechanism to attach authorization and policy to a subsection of the cluster.

Use of multiple namespaces is optional.

This example demonstrates how to use Kubernetes namespaces to subdivide your cluster.

## Before you begin

You need to have a Kubernetes cluster, and the `kubectl` command-line tool must be configured to communicate with your cluster. If you do not already have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:

- [Katacoda](#)
- [Play with Kubernetes](#)

To check the version, enter `kubectl version`.

## Prerequisites

*This example assumes the following:*

1. You have an [existing Kubernetes cluster](#).
2. You have a basic understanding of Kubernetes [Pods](#), [Services](#), and [Deployments](#).

## **Understand the default namespace**

*By default, a Kubernetes cluster will instantiate a default namespace when provisioning the cluster to hold the default set of Pods, Services, and Deployments used by the cluster.*

*Assuming you have a fresh cluster, you can inspect the available namespaces by doing the following:*

```
kubectl get namespaces
```

NAME	STATUS	AGE
default	Active	13m

## **Create new namespaces**

*For this exercise, we will create two additional Kubernetes namespaces to hold our content.*

*Let's imagine a scenario where an organization is using a shared Kubernetes cluster for development and production use cases.*

*The development team would like to maintain a space in the cluster where they can get a view on the list of Pods, Services, and Deployments they use to build and run their application. In this space, Kubernetes resources come and go, and the restrictions on who can or cannot modify resources are relaxed to enable agile development.*

*The operations team would like to maintain a space in the cluster where they can enforce strict procedures on who can or cannot manipulate the set of Pods, Services, and Deployments that run the production site.*

*One pattern this organization could follow is to partition the Kubernetes cluster into two namespaces: *development* and *production*.*

*Let's create two new namespaces to hold our work.*

*Use the file [namespace-dev.json](#) which describes a development namespace:*

[admin/namespace-dev.json](#)



```
{  
  "apiVersion": "v1",  
  "kind": "Namespace",  
  "metadata": {  
    "name": "development",  
    "labels": {  
      "name": "development"  
    }  
  }  
}
```

*Create the development namespace using kubectl.*

```
kubectl create -f https://k8s.io/examples/admin/namespace-dev.json
```

*Save the following contents into file [namespace-prod.json](#) which describes a production namespace:*

[admin/namespace-prod.json](#)



```
{  
  "apiVersion": "v1",  
  "kind": "Namespace",  
  "metadata": {  
    "name": "production",  
    "labels": {  
      "name": "production"  
    }  
  }  
}
```

*And then let's create the production namespace using kubectl.*

```
kubectl create -f https://k8s.io/examples/admin/namespace-prod.json
```

To be sure things are right, let's list all of the namespaces in our cluster.

```
kubectl get namespaces --show-labels
```

NAME	STATUS	AGE	LABELS
default	Active	32m	<none>
development	Active	29s	name=development
production	Active	23s	name=production

## Create pods in each namespace

A Kubernetes namespace provides the scope for Pods, Services, and Deployments in the cluster.

Users interacting with one namespace do not see the content in another namespace.

To demonstrate this, let's spin up a simple Deployment and Pods in the development namespace.

We first check what is the current context:

```
kubectl config view
```

```
apiVersion: v1
clusters:
- cluster:
  certificate-authority-data: REDACTED
  server: https://130.211.122.180
  name: lithe-cocoa-92103_kubernetes
contexts:
- context:
  cluster: lithe-cocoa-92103_kubernetes
  user: lithe-cocoa-92103_kubernetes
  name: lithe-cocoa-92103_kubernetes
current-context: lithe-cocoa-92103_kubernetes
kind: Config
preferences: {}
users:
- name: lithe-cocoa-92103_kubernetes
  user:
    client-certificate-data: REDACTED
    client-key-data: REDACTED
    token: 65rZW78y8HbwXXtSXuUw9DbP4FLjHi4b
```

```
- name: lithe-cocoa-92103_kubernetes-basic-auth
  user:
    password: h5M0FtUUIflBSdI7
    username: admin
```

```
kubectl config current-context
```

```
lithe-cocoa-92103_kubernetes
```

The next step is to define a context for the kubectl client to work in each namespace. The value of "cluster" and "user" fields are copied from the current context.

```
kubectl config set-context dev --namespace=development \
--cluster=lithe-cocoa-92103_kubernetes \
--user=lithe-cocoa-92103_kubernetes
```

```
kubectl config set-context prod --namespace=production \
--cluster=lithe-cocoa-92103_kubernetes \
--user=lithe-cocoa-92103_kubernetes
```

By default, the above commands adds two contexts that are saved into file `.kube/config`. You can now view the contexts and alternate against the two new request contexts depending on which namespace you wish to work against.

To view the new contexts:

```
kubectl config view
```

```
apiVersion: v1
clusters:
- cluster:
  certificate-authority-data: REDACTED
  server: https://130.211.122.180
  name: lithe-cocoa-92103_kubernetes
contexts:
- context:
  cluster: lithe-cocoa-92103_kubernetes
  user: lithe-cocoa-92103_kubernetes
  name: lithe-cocoa-92103_kubernetes
- context:
  cluster: lithe-cocoa-92103_kubernetes
  namespace: development
  user: lithe-cocoa-92103_kubernetes
  name: dev
```

```
- context:
  cluster: lithe-cocoa-92103_kubernetes
  namespace: production
  user: lithe-cocoa-92103_kubernetes
  name: prod
current-context: lithe-cocoa-92103_kubernetes
kind: Config
preferences: {}
users:
- name: lithe-cocoa-92103_kubernetes
  user:
    client-certificate-data: REDACTED
    client-key-data: REDACTED
    token: 65rZW78y8HbwXXtSXuUw9DbP4FLjHi4b
- name: lithe-cocoa-92103_kubernetes-basic-auth
  user:
    password: h5M0FtUUIf1BSdI7
    username: admin
```

*Let's switch to operate in the development namespace.*

```
kubectl config use-context dev
```

*You can verify your current context by doing the following:*

```
kubectl config current-context
```

```
dev
```

*At this point, all requests we make to the Kubernetes cluster from the command line are scoped to the development namespace.*

*Let's create some contents.*

[admin/snowflake-deployment.yaml](#)  


```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: snowflake
    name: snowflake
spec:
  replicas: 2
  selector:
```

```

matchLabels:
  app: snowflake
template:
  metadata:
    labels:
      app: snowflake
  spec:
    containers:
      - image: k8s.gcr.io/serve_hostname
        imagePullPolicy: Always
        name: snowflake

```

*Apply the manifest to create a Deployment*

```
kubectl apply -f https://k8s.io/examples/admin/snowflake-deployment.yaml
```

*We have just created a deployment whose replica size is 2 that is running the pod called snowflake with a basic container that just serves the hostname.*

```
kubectl get deployment
```

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
snowflake	2/2	2	2	2m

```
kubectl get pods -l app=snowflake
```

NAME	READY	STATUS	RESTARTS	AGE
snowflake-3968820950-9dgr8	1/1	Running	0	2m
snowflake-3968820950-vgc4n	1/1	Running	0	2m

*And this is great, developers are able to do what they want, and they do not have to worry about affecting content in the production namespace.*

*Let's switch to the production namespace and show how resources in one namespace are hidden from the other.*

```
kubectl config use-context prod
```

*The production namespace should be empty, and the following commands should return nothing.*

```
kubectl get deployment  
kubectl get pods
```

Production likes to run cattle, so let's create some cattle pods.

```
kubectl create deployment cattle --image=k8s.gcr.io/  
serve_hostname --replicas=5
```

```
kubectl get deployment
```

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
cattle	5/5	5	5	10s

```
kubectl get pods -l app=cattle
```

NAME	READY	STATUS	RESTARTS	AGE
cattle-2263376956-41xy6	1/1	Running	0	34s
cattle-2263376956-kw466	1/1	Running	0	34s
cattle-2263376956-n4v97	1/1	Running	0	34s
cattle-2263376956-p5p3i	1/1	Running	0	34s
cattle-2263376956-sxpth	1/1	Running	0	34s

At this point, it should be clear that the resources users create in one namespace are hidden from the other namespace.

As the policy support in Kubernetes evolves, we will extend this scenario to show how you can provide different authorization rules for each 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](#).

Last modified July 27, 2020 at 4:18 AM PST: [Revise Pod concept \(#22603\) \(49eee8fd3\)](#)

[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Prerequisites](#)
- [Understand the default namespace](#)
- [Create new namespaces](#)

- [Create pods in each namespace](#)

# Operating etcd clusters for Kubernetes

*etcd* is a consistent and highly-available key value store used as Kubernetes' backing store for all cluster data.

If your Kubernetes cluster uses etcd as its backing store, make sure you have a [back up](#) plan for those data.

You can find in-depth information about etcd in the official [documentation](#).

## Before you begin

You need to have a Kubernetes cluster, and the `kubectl` command-line tool must be configured to communicate with your cluster. If you do not already have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:

- [Katacoda](#)
- [Play with Kubernetes](#)

To check the version, enter `kubectl version`.

## Prerequisites

- Run etcd as a cluster of odd members.
- etcd is a leader-based distributed system. Ensure that the leader periodically send heartbeats on time to all followers to keep the cluster stable.
- Ensure that no resource starvation occurs.

Performance and stability of the cluster is sensitive to network and disk IO. Any resource starvation can lead to heartbeat timeout, causing instability of the cluster. An unstable etcd indicates that no leader is elected. Under such circumstances, a cluster cannot make any changes to its current state, which implies no new pods can be scheduled.

- Keeping stable etcd clusters is critical to the stability of Kubernetes clusters. Therefore, run etcd clusters on dedicated machines or isolated environments for [guaranteed resource requirements](#).
- The minimum recommended version of etcd to run in production is 3.2 .10+.

# **Resource requirements**

*Operating etcd with limited resources is suitable only for testing purposes. For deploying in production, advanced hardware configuration is required. Before deploying etcd in production, see [resource requirement reference documentation](#).*

## **Starting etcd clusters**

*This section covers starting a single-node and multi-node etcd cluster.*

### **Single-node etcd cluster**

*Use a single-node etcd cluster only for testing purpose.*

1. Run the following:

```
./etcd --listen-client-urls=http://$PRIVATE_IP:2379 --  
advertise-client-urls=http://$PRIVATE_IP:2379
```

2. Start Kubernetes API server with the flag `--etcd-servers=$PRIVATE_IP:2379`.

*Replace `PRIVATE_IP` with your etcd client IP.*

### **Multi-node etcd cluster**

*For durability and high availability, run etcd as a multi-node cluster in production and back it up periodically. A five-member cluster is recommended in production. For more information, see [FAQ Documentation](#).*

*Configure an etcd cluster either by static member information or by dynamic discovery. For more information on clustering, see [etcd Clustering Documentation](#).*

*For an example, consider a five-member etcd cluster running with the following client URLs: `http://$IP1:2379`, `http://$IP2:2379`, `http://$IP3:2379`, `http://$IP4:2379`, and `http://$IP5:2379`. To start a Kubernetes API server:*

1. Run the following:

```
./etcd --listen-client-urls=http://$IP1:2379, http://$IP2:2379, http://$IP3:2379, http://$IP4:2379, http://$IP5:2379 --advertise-client-urls=http://$IP1:2379, http://$IP2:2379, http://$IP3:2379, http://$IP4:2379, http://$IP5:2379
```

2. Start Kubernetes API servers with the flag `--etcd-servers=$IP1:2379, $IP2:2379, $IP3:2379, $IP4:2379, $IP5:2379`.

Replace *IP* with your client IP addresses.

## **Multi-node etcd cluster with load balancer**

To run a load balancing etcd cluster:

1. Set up an etcd cluster.
2. Configure a load balancer in front of the etcd cluster. For example, let the address of the load balancer be `$LB`.
3. Start Kubernetes API Servers with the flag `--etcd-servers=$LB:2379`.

## **Securing etcd clusters**

Access to etcd is equivalent to root permission in the cluster so ideally only the API server should have access to it. Considering the sensitivity of the data, it is recommended to grant permission to only those nodes that require access to etcd clusters.

To secure etcd, either set up firewall rules or use the security features provided by etcd. etcd security features depend on x509 Public Key Infrastructure (PKI). To begin, establish secure communication channels by generating a key and certificate pair. For example, use key pairs `peer.key` and `peer.cert` for securing communication between etcd members, and `client.key` and `client.cert` for securing communication between etcd and its clients. See the [example scripts](#) provided by the etcd project to generate key pairs and CA files for client authentication.

## **Securing communication**

To configure etcd with secure peer communication, specify flags `--peer-key-file=peer.key` and `--peer-cert-file=peer.cert`, and use https as URL schema.

*Similarly, to configure etcd with secure client communication, specify flags -key-file=k8sclient.key and --cert-file=k8sclient.cert, and use https as URL schema.*

## ***Limiting access of etcd clusters***

*After configuring secure communication, restrict the access of etcd cluster to only the Kubernetes API server. Use TLS authentication to do so.*

*For example, consider key pairs k8sclient.key and k8sclient.cert that are trusted by the CA etcd.ca. When etcd is configured with --client-cert-auth along with TLS, it verifies the certificates from clients by using system CAs or the CA passed in by --trusted-ca-file flag. Specifying flags --client-cert-auth=true and --trusted-ca-file=etcd.ca will restrict the access to clients with the certificate k8sclient.cert.*

*Once etcd is configured correctly, only clients with valid certificates can access it. To give Kubernetes API server the access, configure it with the flags --etcd-certfile=k8sclient.cert, --etcd-keyfile=k8sclient.key and --etcd-cafile=ca.cert.*

**Note:** etcd authentication is not currently supported by Kubernetes. For more information, see the related issue [Support Basic Auth for Etcd v2](#).

## ***Replacing a failed etcd member***

*etcd cluster achieves high availability by tolerating minor member failures. However, to improve the overall health of the cluster, replace failed members immediately. When multiple members fail, replace them one by one. Replacing a failed member involves two steps: removing the failed member and adding a new member.*

*Though etcd keeps unique member IDs internally, it is recommended to use a unique name for each member to avoid human errors. For example, consider a three-member etcd cluster. Let the URLs be, member1=http://10.0.0.1, member2=http://10.0.0.2, and member3=http://10.0.0.3. When member1 fails, replace it with member4=http://10.0.0.4.*

1. Get the member ID of the failed member1:

```
etcdctl --endpoints=http://10.0.0.2,http://10.0.0.3 member list
```

*The following message is displayed:*

```
8211f1d0f64f3269, started, member1, http://10.0.0.1:2380,  
http://10.0.0.1:2379  
91bc3c398fb3c146, started, member2, http://10.0.0.2:2380,  
http://10.0.0.2:2379  
fd422379fda50e48, started, member3, http://10.0.0.3:2380,  
http://10.0.0.3:2379
```

2. Remove the failed member:

```
etcdctl member remove 8211f1d0f64f3269
```

*The following message is displayed:*

```
Removed member 8211f1d0f64f3269 from cluster
```

3. Add the new member:

```
./etcdctl member add member4 --peer-urls=http://10.0.0.4:2380
```

*The following message is displayed:*

```
Member 2be1eb8f84b7f63e added to cluster ef37ad9dc622a7c4
```

4. Start the newly added member on a machine with the IP 10.0.0.4:

```
export ETCD_NAME="member4"  
export ETCD_INITIAL_CLUSTER="member2=http://  
10.0.0.2:2380,member3=http://10.0.0.3:2380,member4=http://  
10.0.0.4:2380"  
export ETCD_INITIAL_CLUSTER_STATE=existing  
etcd [flags]
```

5. Do either of the following:

1. Update its `--etcd-servers` flag to make Kubernetes aware of the configuration changes, then restart the Kubernetes API server.
2. Update the load balancer configuration if a load balancer is used in the deployment.

For more information on cluster reconfiguration, see [etcd Reconfiguration Documentation](#).

## **Backing up an etcd cluster**

All Kubernetes objects are stored on etcd. Periodically backing up the etcd cluster data is important to recover Kubernetes clusters under disaster scenarios, such as losing all master nodes. The snapshot file contains all the

*Kubernetes states and critical information. In order to keep the sensitive Kubernetes data safe, encrypt the snapshot files.*

*Backing up an etcd cluster can be accomplished in two ways: etcd built-in snapshot and volume snapshot.*

## **Built-in snapshot**

*etcd supports built-in snapshot, so backing up an etcd cluster is easy. A snapshot may either be taken from a live member with the etcdctl snapshot save command or by copying the member/snap/db file from an etcd [data directory](#) that is not currently used by an etcd process. Taking the snapshot will normally not affect the performance of the member.*

*Below is an example for taking a snapshot of the keyspace served by \$ENDPOINT to the file snapshotdb:*

```
ETCDCTL_API=3 etcdctl --endpoints $ENDPOINT snapshot save
snapshotdb
# exit 0

# verify the snapshot
ETCDCTL_API=3 etcdctl --write-out=table snapshot status
snapshotdb
+-----+-----+-----+-----+
| HASH | REVISION | TOTAL KEYS | TOTAL SIZE |
+-----+-----+-----+-----+
| fe01cf57 |      10 |          7 | 2.1 MB      |
+-----+-----+-----+-----+
```

## **Volume snapshot**

*If etcd is running on a storage volume that supports backup, such as Amazon Elastic Block Store, back up etcd data by taking a snapshot of the storage volume.*

## **Scaling up etcd clusters**

*Scaling up etcd clusters increases availability by trading off performance. Scaling does not increase cluster performance nor capability. A general rule is not to scale up or down etcd clusters. Do not configure any auto scaling groups for etcd clusters. It is highly recommended to always run a static*

*five-member etcd cluster for production Kubernetes clusters at any officially supported scale.*

*A reasonable scaling is to upgrade a three-member cluster to a five-member one, when more reliability is desired. See [etcd Reconfiguration Documentation](#) for information on how to add members into an existing cluster.*

## **Restoring an etcd cluster**

*etcd supports restoring from snapshots that are taken from an etcd process of the [major.minor](#) version. Restoring a version from a different patch version of etcd also is supported. A restore operation is employed to recover the data of a failed cluster.*

*Before starting the restore operation, a snapshot file must be present. It can either be a snapshot file from a previous backup operation, or from a remaining [data directory](#). For more information and examples on restoring a cluster from a snapshot file, see [etcd disaster recovery documentation](#).*

*If the access URLs of the restored cluster is changed from the previous cluster, the Kubernetes API server must be reconfigured accordingly. In this case, restart Kubernetes API server with the flag `--etcd-servers=$NEW_ETCD_CLUSTER` instead of the flag `--etcd-servers=$OLD_ETCD_CLUSTER`. Replace `$NEW_ETCD_CLUSTER` and `$OLD_ETCD_CLUSTER` with the respective IP addresses. If a load balancer is used in front of an etcd cluster, you might need to update the load balancer instead.*

*If the majority of etcd members have permanently failed, the etcd cluster is considered failed. In this scenario, Kubernetes cannot make any changes to its current state. Although the scheduled pods might continue to run, no new pods can be scheduled. In such cases, recover the etcd cluster and potentially reconfigure Kubernetes API server to fix the issue.*

### **Note:**

*If any API servers are running in your cluster, you should not attempt to restore instances of etcd. Instead, follow these steps to restore etcd:*

- stop all kube-apiserver instances
- restore state in all etcd instances
- restart all kube-apiserver instances

*We also recommend restarting any components (e.g. kube-scheduler, kube-controller-manager, kubelet) to ensure that they*

*don't rely on some stale data. Note that in practice, the restore takes a bit of time. During the restoration, critical components will lose leader lock and restart themselves.*

## **Upgrading and rolling back etcd clusters**

*As of Kubernetes v1.13.0, etcd2 is no longer supported as a storage backend for new or existing Kubernetes clusters. The timeline for Kubernetes support for etcd2 and etcd3 is as follows:*

- *Kubernetes v1.0: etcd2 only*
- *Kubernetes v1.5.1: etcd3 support added, new clusters still default to etcd2*
- *Kubernetes v1.6.0: new clusters created with `kube-up.sh` default to etcd3, and `kube-apiserver` defaults to etcd3*
- *Kubernetes v1.9.0: deprecation of etcd2 storage backend announced*
- *Kubernetes v1.13.0: etcd2 storage backend removed, `kube-apiserver` will refuse to start with `--storage-backend=etcd2`, with the message etcd2 is no longer a supported storage backend*

*Before upgrading a v1.12.x `kube-apiserver` using `--storage-backend=etcd2` to v1.13.x, etcd v2 data must be migrated to the v3 storage backend and `kube-apiserver` invocations must be changed to use `--storage-backend=etcd3`.*

*The process for migrating from etcd2 to etcd3 is highly dependent on how the etcd cluster was deployed and configured, as well as how the Kubernetes cluster was deployed and configured. We recommend that you consult your cluster provider's documentation to see if there is a predefined solution.*

*If your cluster was created via `kube-up.sh` and is still using etcd2 as its storage backend, please consult the [Kubernetes v1.12 etcd cluster upgrade docs](#)*

## **Known issue: etcd client balancer with secure endpoints**

*The etcd v3 client, released in etcd v3.3.13 or earlier, has a [critical bug](#) which affects the `kube-apiserver` and HA deployments. The etcd client balancer failover does not properly work against secure endpoints. As a result, etcd servers may fail or disconnect briefly from the `kube-apiserver`. This affects `kube-apiserver` HA deployments.*

The fix was made in [etcd v3.4](#) (and backported to v3.3.14 or later): the new client now creates its own credential bundle to correctly set authority target in dial function.

Because the fix requires gRPC dependency upgrade (to v1.23.0), downstream Kubernetes [did not backport etcd upgrades](#). Which means the [etcd fix in kube-apiserver](#) is only available from Kubernetes 1.16.

To urgently fix this bug for Kubernetes 1.15 or earlier, build a custom kube-apiserver. You can make local changes to [vendor/google.golang.org/grpc/credentials/credentials.go](#) with [etcd@db61ee106](#).

See ["kube-apiserver 1.13.x refuses to work when first etcd-server is not available"](#).

## 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](#).

Last modified November 05, 2020 at 11:05 AM PST: [document one should restart all system components after restoring etcd \(c6175427b\)](#)  
[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Prerequisites](#)
- [Resource requirements](#)
- [Starting etcd clusters](#)
  - [Single-node etcd cluster](#)
  - [Multi-node etcd cluster](#)
  - [Multi-node etcd cluster with load balancer](#)
- [Securing etcd clusters](#)
  - [Securing communication](#)
  - [Limiting access of etcd clusters](#)
- [Replacing a failed etcd member](#)
- [Backing up an etcd cluster](#)
  - [Built-in snapshot](#)
  - [Volume snapshot](#)
- [Scaling up etcd clusters](#)
- [Restoring an etcd cluster](#)
- [Upgrading and rolling back etcd clusters](#)
- [Known issue: etcd client balancer with secure endpoints](#)

# **Reconfigure a Node's Kubelet in a Live Cluster**

**FEATURE STATE:** Kubernetes v1.11 [beta]

[Dynamic Kubelet Configuration](#) allows you to change the configuration of each [kubelet](#) in a running Kubernetes cluster, by deploying a [ConfigMap](#) and configuring each [Node](#) to use it.

**Warning:** All kubelet configuration parameters can be changed dynamically, but this is unsafe for some parameters. Before deciding to change a parameter dynamically, you need a strong understanding of how that change will affect your cluster's behavior. Always carefully test configuration changes on a small set of nodes before rolling them out cluster-wide. Advice on configuring specific fields is available in the inline [Kubelet Configuration type documentation](#).

## **Before you begin**

You need to have a Kubernetes cluster. You also need `kubectl` v1.11 or higher, configured to communicate with your cluster. Your Kubernetes server must be at or later than version v1.11. To check the version, enter `kubectl version`. Your cluster API server version (eg v1.12) must be no more than one minor version away from the version of `kubectl` that you are using. For example, if your cluster is running v1.16 then you can use `kubectl` v1.15, v1.16 or v1.17; other combinations [aren't supported](#).

Some of the examples use the command line tool [jq](#). You do not need `jq` to complete the task, because there are manual alternatives.

For each node that you're reconfiguring, you must set the `kubelet --dynamic-config-dir` flag to a writable directory.

## **Reconfiguring the kubelet on a running node in your cluster**

### **Basic workflow overview**

The basic workflow for configuring a kubelet in a live cluster is as follows:

1. Write a YAML or JSON configuration file containing the kubelet's configuration.
2. Wrap this file in a ConfigMap and save it to the Kubernetes control plane.
3. Update the kubelet's corresponding Node object to use this ConfigMap.

*Each kubelet watches a configuration reference on its respective Node object. When this reference changes, the kubelet downloads the new configuration, updates a local reference to refer to the file, and exits. For the feature to work correctly, you must be running an OS-level service manager (such as systemd), which will restart the kubelet if it exits. When the kubelet is restarted, it will begin using the new configuration.*

*The new configuration completely overrides configuration provided by --config, and is overridden by command-line flags. Unspecified values in the new configuration will receive default values appropriate to the configuration version (e.g. kubelet.config.k8s.io/v1beta1), unless overridden by flags.*

*The status of the Node's kubelet configuration is reported via Node.Status.Config. Once you have updated a Node to use the new ConfigMap, you can observe this status to confirm that the Node is using the intended configuration.*

*This document describes editing Nodes using kubectl edit. There are other ways to modify a Node's spec, including kubectl patch, for example, which facilitate scripted workflows.*

*This document only describes a single Node consuming each ConfigMap. Keep in mind that it is also valid for multiple Nodes to consume the same ConfigMap.*

**Warning:** While it is possible to change the configuration by updating the ConfigMap in-place, this causes all kubelets configured with that ConfigMap to update simultaneously. It is much safer to treat ConfigMaps as immutable by convention, aided by kubectl's --append-hash option, and incrementally roll out updates to Node.Spec.ConfigSource.

## **Automatic RBAC rules for Node Authorizer**

*Previously, you were required to manually create RBAC rules to allow Nodes to access their assigned ConfigMaps. The Node Authorizer now automatically configures these rules.*

## **Generating a file that contains the current configuration**

*The Dynamic Kubelet Configuration feature allows you to provide an override for the entire configuration object, rather than a per-field overlay. This is a simpler model that makes it easier to trace the source of configuration values and debug issues. The compromise, however, is that you must start with knowledge of the existing configuration to ensure that you only change the fields you intend to change.*

*The kubelet loads settings from its configuration file, but you can set command line flags to override the configuration in the file. This means that if you only know the contents of the configuration file, and you don't know*

*the command line overrides, then you do not know the running configuration either.*

*Because you need to know the running configuration in order to override it, you can fetch the running configuration from the kubelet. You can generate a config file containing a Node's current configuration by accessing the kubelet's configz endpoint, through kubectl proxy. The next section explains how to do this.*

**Caution:** The kubelet's configz endpoint is there to help with debugging, and is not a stable part of kubelet behavior. Do not rely on the behavior of this endpoint for production scenarios or for use with automated tools.

For more information on configuring the kubelet via a configuration file, see [Set kubelet parameters via a config file](#).

## Generate the configuration file

**Note:** The steps below use the jq command to streamline working with JSON. To follow the tasks as written, you need to have jq installed. You can adapt the steps if you prefer to extract the kubel etconfig subobject manually.

1. Choose a Node to reconfigure. In this example, the name of this Node is referred to as `NODE_NAME`.
2. Start the kubectl proxy in the background using the following command:

```
kubectl proxy --port=8001 &
```

3. Run the following command to download and unpack the configuration from the configz endpoint. The command is long, so be careful when copying and pasting. **If you use zsh**, note that common zsh configurations add backslashes to escape the opening and closing curly braces around the variable name in the URL. For example: `$ {NODE_NAME}` will be rewritten as `\$\\{NODE_NAME\\}` during the paste. You must remove the backslashes before running the command, or the command will fail.

```
NODE_NAME="the-name-of-the-node-you-are-reconfiguring"; curl -SSL "http://localhost:8001/api/v1/nodes/${NODE_NAME}/proxy/configz" | jq '.kubeletconfig|.kind="KubeletConfiguration"|.apiVersion="kubelet.config.k8s.io/v1beta1"' > kubelet_configz_${NODE_NAME}
```

**Note:** You need to manually add the kind and apiVersion to the downloaded object, because those fields are not reported by the configz endpoint.

## **Edit the configuration file**

Using a text editor, change one of the parameters in the file generated by the previous procedure. For example, you might edit the parameter `eventRecordingQPS`, that controls rate limiting for event recording.

## **Push the configuration file to the control plane**

Push the edited configuration file to the control plane with the following command:

```
kubectl -n kube-system create configmap my-node-config --from-file=kubelet=kubelet_configz_${NODE_NAME} --append-hash -o yaml
```

This is an example of a valid response:

```
apiVersion: v1
kind: ConfigMap
metadata:
  creationTimestamp: 2017-09-14T20:23:33Z
  name: my-node-config-gkt4c2m4b2
  namespace: kube-system
  resourceVersion: "119980"
  uid: 946d785e-998a-11e7-a8dd-42010a800006
data:
  kubelet: |
    { ... }
```

You created that `ConfigMap` inside the `kube-system` namespace because the `kubelet` is a Kubernetes system component.

The `--append-hash` option appends a short checksum of the `ConfigMap` contents to the name. This is convenient for an edit-then-push workflow, because it automatically, yet deterministically, generates new names for new resources. The name that includes this generated hash is referred to as `CONFIG_MAP_NAME` in the following examples.

## **Set the Node to use the new configuration**

Edit the Node's reference to point to the new `ConfigMap` with the following command:

```
kubectl edit node ${NODE_NAME}
```

In your text editor, add the following YAML under `spec`:

```
configSource:
  configMap:
    name: CONFIG_MAP_NAME # replace CONFIG_MAP_NAME with the
    name of the ConfigMap
    namespace: kube-system
    kubeletConfigKey: kubelet
```

You must specify all three of `name`, `namespace`, and `kubeletConfigKey`. The `kubeletConfigKey` parameter shows the kubelet which key of the ConfigMap contains its config.

### Observe that the Node begins using the new configuration

Retrieve the Node using the `kubectl get node ${NODE_NAME} -o yaml` command and inspect `Node.Status.Config`. The config sources corresponding to the `active`, `assigned`, and `lastKnownGood` configurations are reported in the status.

- The `active` configuration is the version the kubelet is currently running with.
- The `assigned` configuration is the latest version the kubelet has resolved based on `Node.Spec.ConfigSource`.
- The `lastKnownGood` configuration is the version the kubelet will fall back to if an invalid config is assigned in `Node.Spec.ConfigSource`.

The `lastKnownGood` configuration might not be present if it is set to its default value, the local config deployed with the node. The status will update `lastKnownGood` to match a valid `assigned` config after the kubelet becomes comfortable with the config. The details of how the kubelet determines a config should become the `lastKnownGood` are not guaranteed by the API, but is currently implemented as a 10-minute grace period.

You can use the following command (using `jq`) to filter down to the config status:

```
kubectl get no ${NODE_NAME} -o json | jq '.status.config'
```

The following is an example response:

```
{
  "active": {
    "configMap": {
      "kubeletConfigKey": "kubelet",
      "name": "my-node-config-9mbkccg2cc",
      "namespace": "kube-system",
      "resourceVersion": "1326",
      "uid": "705ab4f5-6393-11e8-b7cc-42010a800002"
    }
  },
  "assigned": {
    "configMap": {
      "kubeletConfigKey": "kubelet",
      "name": "my-node-config-9mbkccg2cc",
      "namespace": "kube-system",
      "resourceVersion": "1326",
      "uid": "705ab4f5-6393-11e8-b7cc-42010a800002"
    }
  },
  "lastKnownGood": {
```

```

    "configMap": {
      "kubeletConfigKey": "kubelet",
      "name": "my-node-config-9mbkccg2cc",
      "namespace": "kube-system",
      "resourceVersion": "1326",
      "uid": "705ab4f5-6393-11e8-b7cc-42010a800002"
    }
  }
}

```

(if you do not have jq, you can look at the whole response and find `Node.Status.Config` by eye).

If an error occurs, the kubelet reports it in the `Node.Status.Config.Error` structure. Possible errors are listed in [Understanding Node.Status.Config.Error messages](#). You can search for the identical text in the kubelet log for additional details and context about the error.

## **Make more changes**

Follow the workflow above to make more changes and push them again. Each time you push a `ConfigMap` with new contents, the `--append-hash` `kubectl` option creates the `ConfigMap` with a new name. The safest rollout strategy is to first create a new `ConfigMap`, and then update the `Node` to use the new `ConfigMap`.

### **Reset the Node to use its local default configuration**

To reset the `Node` to use the configuration it was provisioned with, edit the `Node` using `kubectl edit node ${NODE_NAME}` and remove the `Node.Spec.ConfigSource` field.

### **Observe that the Node is using its local default configuration**

After removing this subfield, `Node.Status.Config` eventually becomes empty, since all config sources have been reset to nil, which indicates that the local default config is assigned, active, and `lastKnownGood`, and no error is reported.

## **kubectl patch example**

You can change a `Node`'s `configSource` using several different mechanisms. This example uses `kubectl patch`:

```

kubectl patch node ${NODE_NAME} -p "{\"spec\":{\"configSource\":
{\\"configMap\\\":{\\\"name\\\":\\\"${CONFIG_MAP_NAME}\\\",\\\"namespace\\\":
\\\"kube-system\\\",\\\"kubeletConfigKey\\\":\\\"kubelet\\\"}}}}"

```

## **Understanding how the kubelet checkpoints config**

*When a new config is assigned to the Node, the kubelet downloads and unpacks the config payload as a set of files on the local disk. The kubelet also records metadata that locally tracks the assigned and last-known-good config sources, so that the kubelet knows which config to use across restarts, even if the API server becomes unavailable. After checkpointing a config and the relevant metadata, the kubelet exits if it detects that the assigned config has changed. When the kubelet is restarted by the OS-level service manager (such as `systemd`), it reads the new metadata and uses the new config.*

*The recorded metadata is fully resolved, meaning that it contains all necessary information to choose a specific config version - typically a `UID` and `ResourceVersion`. This is in contrast to `Node.Spec.ConfigSource`, where the intended config is declared via the idempotent `namespace/name` that identifies the target `ConfigMap`; the kubelet tries to use the latest version of this `ConfigMap`.*

*When you are debugging problems on a node, you can inspect the kubelet's config metadata and checkpoints. The structure of the kubelet's checkpointing directory is:*

```
- --dynamic-config-dir (root for managing dynamic config)
| - meta
  | - assigned (encoded kubeletconfig/
v1beta1.SerializedNodeConfigSource object, indicating the
assigned config)
  | - last-known-good (encoded kubeletconfig/
v1beta1.SerializedNodeConfigSource object, indicating the last-
known-good config)
| - checkpoints
  | - uid1 (dir for versions of object identified by uid1)
    | - resourceVersion1 (dir for unpacked files from
resourceVersion1 of object with uid1)
    | - ...
  | - ...
```

## **Understanding Node.Status.Config.Error messages**

*The following table describes error messages that can occur when using Dynamic Kubelet Config. You can search for the identical text in the Kubelet log for additional details and context about the error.*

Error Message	Possible Causes
failed to load config, see Kubelet log for details	The kubelet likely could not parse the downloaded config payload, or encountered a filesystem error attempting to load the payload from disk.
failed to validate config, see Kubelet log for details	The configuration in the payload, combined with any command-line flag overrides, and the sum of feature gates from flags, the config file, and the remote payload, was determined to be invalid by the kubelet.
invalid NodeConfigSource, exactly one subfield must be non-nil, but all were nil	Since Node.Spec.ConfigSource is validated by the API server to contain at least one non-nil subfield, this likely means that the kubelet is older than the API server and does not recognize a newer source type.
failed to sync: failed to download config, see Kubelet log for details	The kubelet could not download the config. It is possible that Node.Spec.ConfigSource could not be resolved to a concrete API object, or that network errors disrupted the download attempt. The kubelet will retry the download when in this error state.
failed to sync: internal failure, see Kubelet log for details	The kubelet encountered some internal problem and failed to update its config as a result. Examples include filesystem errors and reading objects from the internal informer cache.
internal failure, see Kubelet log for details	The kubelet encountered some internal problem while manipulating config, outside of the configuration sync loop.

## What's next

- For more information on configuring the kubelet via a configuration file, see [Set kubelet parameters via a config file](#).
- See the reference documentation for [NodeConfigSource](#)

## 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](#).

Last modified July 05, 2020 at 10:42 PM PST: [Fix issue of document link in some documents \(64f62106c\)](#)

[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)

- [Reconfiguring the kubelet on a running node in your cluster](#)
  - [Basic workflow overview](#)
  - [Automatic RBAC rules for Node Authorizer](#)
  - [Generating a file that contains the current configuration](#)
- [kubectl patch example](#)
- [Understanding how the kubelet checkpoints config](#)
- [Understanding Node.Status.Config.Error messages](#)
- [What's next](#)

# **Reserve Compute Resources for System Daemons**

*Kubernetes nodes can be scheduled to Capacity. Pods can consume all the available capacity on a node by default. This is an issue because nodes typically run quite a few system daemons that power the OS and Kubernetes itself. Unless resources are set aside for these system daemons, pods and system daemons compete for resources and lead to resource starvation issues on the node.*

*The kubelet exposes a feature named `Node Allocatable` that helps to reserve compute resources for system daemons. Kubernetes recommends cluster administrators to configure `Node Allocatable` based on their workload density on each node.*

## **Before you begin**

*You need to have a Kubernetes cluster, and the `kubectl` command-line tool must be configured to communicate with your cluster. If you do not already have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:*

- [Katacoda](#)
- [Play with Kubernetes](#)

*Your Kubernetes server must be at or later than version 1.8. To check the version, enter `kubectl version`. Your Kubernetes server must be at or later than version 1.17 to use the kubelet command line option `--reserved-cpus` to set an [explicitly reserved CPU list](#).*

## **Node Allocatable**

*Allocatable on a Kubernetes node is defined as the amount of compute resources that are available for pods. The scheduler does not over-subscribe Allocatable. CPU, memory and ephemeral-storage are supported as of now.*

*Node Allocatable is exposed as part of v1.Node object in the API and as part of kubectl describe node in the CLI.*

*Resources can be reserved for two categories of system daemons in the kubelet.*

## ***Enabling QoS and Pod level cgroups***

*To properly enforce node allocatable constraints on the node, you must enable the new cgroup hierarchy via the --cgroups-per-qos flag. This flag is enabled by default. When enabled, the kubelet will parent all end-user pods under a cgroup hierarchy managed by the kubelet.*

## ***Configuring a cgroup driver***

The `kubelet` supports manipulation of the cgroup hierarchy on the host using a cgroup driver. The driver is configured via the `--cgroup-driver` flag.

The supported values are the following:

- `cgroupfs` is the default driver that performs direct manipulation of the cgroup filesystem on the host in order to manage cgroup sandboxes.
- `systemd` is an alternative driver that manages cgroup sandboxes using transient slices for resources that are supported by that init system.

Depending on the configuration of the associated container runtime, operators may have to choose a particular cgroup driver to ensure proper system behavior. For example, if operators use the `systemd` cgroup driver provided by the `docker` runtime, the `kubelet` must be configured to use the `systemd` cgroup driver.

## Kube Reserved

- **Kubelet Flag:** `--kube-reserved=[cpu=100m][,][memory=100Mi][,][ephemeral-storage=1Gi][,][pid=1000]`
- **Kubelet Flag:** `--kube-reserved-cgroup=`

`kube-reserved` is meant to capture resource reservation for kubernetes system daemons like the `kubelet`, `container runtime`, `node problem detector`, etc. It is not meant to reserve resources for system daemons that are run as pods. `kube-reserved` is typically a function of pod density on the nodes.

In addition to `cpu`, `memory`, and `ephemeral-storage`, `pid` may be specified to reserve the specified number of process IDs for kubernetes system daemons.

To optionally enforce `kube-reserved` on system daemons, specify the parent control group for kube daemons as the value for `--kube-reserved-cgroup` kubelet flag.

It is recommended that the kubernetes system daemons are placed under a top level control group (`runtime.slice` on `systemd` machines for example). Each system daemon should ideally run within its own child control group. Refer to [this doc](#) for more details on recommended control group hierarchy.

Note that Kubelet **does not** create `--kube-reserved-cgroup` if it doesn't exist. Kubelet will fail if an invalid cgroup is specified.

## **System Reserved**

- **Kubelet Flag:** `--system-reserved=[cpu=100m][,][memory=100Mi][,][ephemeral-storage=1Gi][,][pid=1000]`
- **Kubelet Flag:** `--system-reserved-cgroup=`

*system-reserved is meant to capture resource reservation for OS system daemons like sshd, udev, etc. system-reserved should reserve memory for the kernel too since kernel memory is not accounted to pods in Kubernetes at this time. Reserving resources for user login sessions is also recommended (user.slice in systemd world).*

*In addition to cpu, memory, and ephemeral-storage, pid may be specified to reserve the specified number of process IDs for OS system daemons.*

*To optionally enforce system-reserved on system daemons, specify the parent control group for OS system daemons as the value for --system-reserved-cgroup kubelet flag.*

*It is recommended that the OS system daemons are placed under a top level control group (system.slice on systemd machines for example).*

*Note that Kubelet **does not** create --system-reserved-cgroup if it doesn't exist. Kubelet will fail if an invalid cgroup is specified.*

## **Explicitly Reserved CPU List**

**FEATURE STATE:** Kubernetes v1.17 [stable]

- **Kubelet Flag:** `--reserved-cpus=0-3`

*reserved-cpus is meant to define an explicit CPU set for OS system daemons and kubernetes system daemons. reserved-cpus is for systems that do not intend to define separate top level cgroups for OS system daemons and kubernetes system daemons with regard to cpuset resource. If the Kubelet **does not** have --system-reserved-cgroup and --kube-reserved-cgroup, the explicit cpuset provided by reserved-cpus will take precedence over the CPUs defined by --kube-reserved and --system-reserved options.*

*This option is specifically designed for Telco/NFV use cases where uncontrolled interrupts/timers may impact the workload performance. you can use this option to define the explicit cpuset for the system/kubernetes daemons as well as the interrupts/timers, so the rest CPUs on the system*

*can be used exclusively for workloads, with less impact from uncontrolled interrupts/timers. To move the system daemon, kubernetes daemons and interrupts/timers to the explicit cpuset defined by this option, other mechanism outside Kubernetes should be used. For example: in Centos, you can do this using the tuned toolset.*

## **Eviction Thresholds**

- **Kubelet Flag:** `--eviction-hard=[memory.available<500Mi]`

*Memory pressure at the node level leads to System OOMs which affects the entire node and all pods running on it. Nodes can go offline temporarily until memory has been reclaimed. To avoid (or reduce the probability of) system OOMs kubelet provides [Out of Resource](#) management. Evictions are supported for memory and ephemeral-storage only. By reserving some memory via `--eviction-hard` flag, the kubelet attempts to evict pods whenever memory availability on the node drops below the reserved value. Hypothetically, if system daemons did not exist on a node, pods cannot use more than capacity - `eviction-hard`. For this reason, resources reserved for evictions are not available for pods.*

## **Enforcing Node Allocatable**

- **Kubelet Flag:** `--enforce-node-allocatable=pods[,][system-reserved][,][kube-reserved]`

*The scheduler treats Allocatable as the available capacity for pods.*

*kubelet enforce Allocatable across pods by default. Enforcement is performed by evicting pods whenever the overall usage across all pods exceeds Allocatable. More details on eviction policy can be found [here](#). This enforcement is controlled by specifying pods value to the kubelet flag `-enforce-node-allocatable`.*

*Optionally, kubelet can be made to enforce kube-reserved and system-reserved by specifying kube-reserved & system-reserved values in the same flag. Note that to enforce kube-reserved or system-reserved, `--kube-reserved-cgroup` or `--system-reserved-cgroup` needs to be specified respectively.*

## **General Guidelines**

*System daemons are expected to be treated similar to Guaranteed pods. System daemons can burst within their bounding control groups and this behavior needs to be managed as part of kubernetes deployments. For example, kubelet should have its own control group and share Kube-reserved resources with the container runtime. However, Kubelet cannot burst and use up all available Node resources if kube-reserved is enforced.*

*Be extra careful while enforcing system-reserved reservation since it can lead to critical system services being CPU starved, OOM killed, or unable to fork on the node. The recommendation is to enforce system-reserved only if a user has profiled their nodes exhaustively to come up with precise estimates and is confident in their ability to recover if any process in that group is oom\_killed.*

- To begin with enforce Allocatable on pods.
- Once adequate monitoring and alerting is in place to track kube system daemons, attempt to enforce kube-reserved based on usage heuristics.
- If absolutely necessary, enforce system-reserved over time.

*The resource requirements of kube system daemons may grow over time as more and more features are added. Over time, kubernetes project will attempt to bring down utilization of node system daemons, but that is not a priority as of now. So expect a drop in Allocatable capacity in future releases.*

## **Example Scenario**

*Here is an example to illustrate Node Allocatable computation:*

- Node has 32Gi of memory, 16 CPUs and 100Gi of Storage
- --kube-reserved is set to cpu=1, memory=2Gi, ephemeral-storage=1Gi
- --system-reserved is set to cpu=500m, memory=1Gi, ephemeral-storage=1Gi
- --eviction-hard is set to memory.available<500Mi, nodefs.available<10%

*Under this scenario, Allocatable will be 14.5 CPUs, 28.5Gi of memory and 88Gi of local storage. Scheduler ensures that the total memory requests across all pods on this node does not exceed 28.5Gi and storage doesn't exceed 88Gi. Kubelet evicts pods whenever the overall memory usage across pods exceeds 28.5Gi, or if overall disk usage exceeds 88Gi. If all processes on the node consume as much CPU as they can, pods together cannot consume more than 14.5 CPUs.*

*If kube-reserved and/or system-reserved is not enforced and system daemons exceed their reservation, kubelet evicts pods whenever the overall node memory usage is higher than 31.5Gi or storage is greater than 90Gi*

## 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](#).

Last modified October 12, 2020 at 4:14 PM PST: [Improve diagram for "Reserve Compute Resources for System Daemons"](#) (e9f483808)  
[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Node Allocatable](#)
  - [Enabling QoS and Pod level cgroups](#)
  - [Configuring a cgroup driver](#)
  - [Kube Reserved](#)
  - [System Reserved](#)
  - [Explicitly Reserved CPU List](#)
  - [Eviction Thresholds](#)
  - [Enforcing Node Allocatable](#)
- [General Guidelines](#)
- [Example Scenario](#)

## Safely Drain a Node

This page shows how to safely drain a [node](#), optionally respecting the `PodDisruptionBudget` you have defined.

### Before you begin

Your Kubernetes server must be at or later than version 1.5. To check the version, enter `kubectl version`.

This task also assumes that you have met the following prerequisites:

1. You do not require your applications to be highly available during the node drain, or
2. You have read about the [PodDisruptionBudget](#) concept, and have [configured PodDisruptionBudgets](#) for applications that need them.

## **(Optional) Configure a disruption budget**

To ensure that your workloads remain available during maintenance, you can configure a [PodDisruptionBudget](#).

If availability is important for any applications that run or could run on the node(s) that you are draining, [configure a PodDisruptionBudgets](#) first and then continue following this guide.

## **Use kubectl drain to remove a node from service**

You can use `kubectl drain` to safely evict all of your pods from a node before you perform maintenance on the node (e.g. kernel upgrade, hardware maintenance, etc.). Safe evictions allow the pod's containers to [gracefully terminate](#) and will respect the PodDisruptionBudgets you have specified.

**Note:** By default `kubectl drain` ignores certain system pods on the node that cannot be killed; see the [kubectl drain](#) documentation for more details.

When `kubectl drain` returns successfully, that indicates that all of the pods (except the ones excluded as described in the previous paragraph) have been safely evicted (respecting the desired graceful termination period, and respecting the PodDisruptionBudget you have defined). It is then safe to bring down the node by powering down its physical machine or, if running on a cloud platform, deleting its virtual machine.

First, identify the name of the node you wish to drain. You can list all of the nodes in your cluster with

```
kubectl get nodes
```

Next, tell Kubernetes to drain the node:

```
kubectl drain <node name>
```

Once it returns (without giving an error), you can power down the node (or equivalently, if on a cloud platform, delete the virtual machine backing the node). If you leave the node in the cluster during the maintenance operation, you need to run

```
kubectl uncordon <node name>
```

afterwards to tell Kubernetes that it can resume scheduling new pods onto the node.

## **Draining multiple nodes in parallel**

The `kubectl drain` command should only be issued to a single node at a time. However, you can run multiple `kubectl drain` commands for different

*nodes in parallel, in different terminals or in the background. Multiple drain commands running concurrently will still respect the PodDisruptionBudget you specify.*

*For example, if you have a StatefulSet with three replicas and have set a PodDisruptionBudget for that set specifying minAvailable: 2, kubectl drain only evicts a pod from the StatefulSet if all three replicas pods are ready; if then you issue multiple drain commands in parallel, Kubernetes respects the PodDisruptionBudget and ensure that only 1 (calculated as replicas - minAvailable) Pod is unavailable at any given time. Any drains that would cause the number of ready replicas to fall below the specified budget are blocked.*

## The Eviction API

*If you prefer not to use [kubectl drain](#) (such as to avoid calling to an external command, or to get finer control over the pod eviction process), you can also programmatically cause evictions using the eviction API.*

*You should first be familiar with using [Kubernetes language clients](#) to access the API.*

*The eviction subresource of a Pod can be thought of as a kind of policy-controlled DELETE operation on the Pod itself. To attempt an eviction (more precisely: to attempt to create an Eviction), you POST an attempted operation. Here's an example:*

```
{  
  "apiVersion": "policy/v1beta1",  
  "kind": "Eviction",  
  "metadata": {  
    "name": "quux",  
    "namespace": "default"  
  }  
}
```

*You can attempt an eviction using curl:*

```
curl -v -H 'Content-type: application/json' https://your-cluster-  
api-endpoint.example/api/v1/namespaces/default/pods/quux/  
eviction -d @eviction.json
```

*The API can respond in one of three ways:*

- *If the eviction is granted, then the Pod is deleted just as if you had sent a DELETE request to the Pod's URL and you get back 200 OK.*
- *If the current state of affairs wouldn't allow an eviction by the rules set forth in the budget, you get back 429 Too Many Requests. This is typically used for generic rate limiting of any requests, but here we mean that this request isn't allowed right now but it may be allowed later.*

- If there is some kind of misconfiguration; for example multiple `PodDisruptionBudgets` that refer the same Pod, you get a 500 Internal Server Error response.

For a given eviction request, there are two cases:

- There is no budget that matches this pod. In this case, the server always returns 200 OK.
- There is at least one budget. In this case, any of the three above responses may apply.

## **Stuck evictions**

In some cases, an application may reach a broken state, one where unless you intervene the eviction API will never return anything other than 429 or 500.

For example: this can happen if ReplicaSet is creating Pods for your application but the replacement Pods do not become Ready. You can also see similar symptoms if the last Pod evicted has a very long termination grace period.

In this case, there are two potential solutions:

- Abort or pause the automated operation. Investigate the reason for the stuck application, and restart the automation.
- After a suitably long wait, `DELETE` the Pod from your cluster's control plane, instead of using the eviction API.

Kubernetes does not specify what the behavior should be in this case; it is up to the application owners and cluster owners to establish an agreement on behavior in these cases.

## **What's next**

- Follow steps to protect your application by [configuring a Pod Disruption Budget](#).

## **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](#).

Last modified October 07, 2020 at 7:16 PM PST: [Revise cluster management task \(59dcd57cc\)](#)

- [Before you begin](#)
- [\(Optional\) Configure a disruption budget](#)
- [Use kubectl drain to remove a node from service](#)
- [Draining multiple nodes in parallel](#)
- [The Eviction API](#)
- [Stuck evictions](#)
- [What's next](#)

# Securing a Cluster

*This document covers topics related to protecting a cluster from accidental or malicious access and provides recommendations on overall security.*

## Before you begin

- You need to have a Kubernetes cluster, and the `kubectl` command-line tool must be configured to communicate with your cluster. If you do not already have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:
  - [Katacoda](#)
  - [Play with Kubernetes](#)

To check the version, enter `kubectl version`.

## Controlling access to the Kubernetes API

*As Kubernetes is entirely API driven, controlling and limiting who can access the cluster and what actions they are allowed to perform is the first line of defense.*

### Use Transport Layer Security (TLS) for all API traffic

*Kubernetes expects that all API communication in the cluster is encrypted by default with TLS, and the majority of installation methods will allow the necessary certificates to be created and distributed to the cluster components. Note that some components and installation methods may enable local ports over HTTP and administrators should familiarize themselves with the settings of each component to identify potentially unsecured traffic.*

### API Authentication

*Choose an authentication mechanism for the API servers to use that matches the common access patterns when you install a cluster. For instance, small single user clusters may wish to use a simple certificate or static Bearer token approach. Larger clusters may wish to integrate an existing OIDC or LDAP server that allow users to be subdivided into groups.*

*All API clients must be authenticated, even those that are part of the infrastructure like nodes, proxies, the scheduler, and volume plugins. These clients are typically [service accounts](#) or use x509 client certificates, and they are created automatically at cluster startup or are setup as part of the cluster installation.*

*Consult the [authentication reference document](#) for more information.*

## **API Authorization**

*Once authenticated, every API call is also expected to pass an authorization check. Kubernetes ships an integrated [Role-Based Access Control \(RBAC\)](#) component that matches an incoming user or group to a set of permissions bundled into roles. These permissions combine verbs (get, create, delete) with resources (pods, services, nodes) and can be namespace or cluster scoped. A set of out of the box roles are provided that offer reasonable default separation of responsibility depending on what actions a client might want to perform. It is recommended that you use the [Node](#) and [RBAC](#) authorizers together, in combination with the [NodeRestriction](#) admission plugin.*

*As with authentication, simple and broad roles may be appropriate for smaller clusters, but as more users interact with the cluster, it may become necessary to separate teams into separate namespaces with more limited roles.*

*With authorization, it is important to understand how updates on one object may cause actions in other places. For instance, a user may not be able to create pods directly, but allowing them to create a deployment, which creates pods on their behalf, will let them create those pods indirectly. Likewise, deleting a node from the API will result in the pods scheduled to that node being terminated and recreated on other nodes. The out of the box roles represent a balance between flexibility and the common use cases, but more limited roles should be carefully reviewed to prevent accidental escalation. You can make roles specific to your use case if the out-of-box ones don't meet your needs.*

*Consult the [authorization reference section](#) for more information.*

## **Controlling access to the Kubelet**

*Kubelets expose HTTPS endpoints which grant powerful control over the node and containers. By default Kubelets allow unauthenticated access to this API.*

*Production clusters should enable Kubelet authentication and authorization.*

*Consult the [Kubelet authentication/authorization reference](#) for more information.*

# **Controlling the capabilities of a workload or user at runtime**

*Authorization in Kubernetes is intentionally high level, focused on coarse actions on resources. More powerful controls exist as **policies** to limit by use case how those objects act on the cluster, themselves, and other resources.*

## **Limiting resource usage on a cluster**

[Resource quota](#) limits the number or capacity of resources granted to a namespace. This is most often used to limit the amount of CPU, memory, or persistent disk a namespace can allocate, but can also control how many pods, services, or volumes exist in each namespace.

[Limit ranges](#) restrict the maximum or minimum size of some of the resources above, to prevent users from requesting unreasonably high or low values for commonly reserved resources like memory, or to provide default limits when none are specified.

## **Controlling what privileges containers run with**

A pod definition contains a [security context](#) that allows it to request access to running as a specific Linux user on a node (like root), access to run privileged or access the host network, and other controls that would otherwise allow it to run unfettered on a hosting node. [Pod security policies](#) can limit which users or service accounts can provide dangerous security context settings. For example, pod security policies can limit volume mounts, especially `hostPath`, which are aspects of a pod that should be controlled.

Generally, most application workloads need limited access to host resources so they can successfully run as a root process (`uid 0`) without access to host information. However, considering the privileges associated with the root user, you should write application containers to run as a non-root user. Similarly, administrators who wish to prevent client applications from escaping their containers should use a restrictive pod security policy.

## **Preventing containers from loading unwanted kernel modules**

The Linux kernel automatically loads kernel modules from disk if needed in certain circumstances, such as when a piece of hardware is attached or a filesystem is mounted. Of particular relevance to Kubernetes, even unprivileged processes can cause certain network-protocol-related kernel modules to be loaded, just by creating a socket of the appropriate type. This may allow an attacker to exploit a security hole in a kernel module that the administrator assumed was not in use.

*To prevent specific modules from being automatically loaded, you can uninstall them from the node, or add rules to block them. On most Linux distributions, you can do that by creating a file such as `/etc/modprobe.d/kubernetes-blacklist.conf` with contents like:*

```
# DCCP is unlikely to be needed, has had multiple serious
# vulnerabilities, and is not well-maintained.
blacklist dccp

# SCTP is not used in most Kubernetes clusters, and has also had
# vulnerabilities in the past.
blacklist sctp
```

*To block module loading more generically, you can use a Linux Security Module (such as SELinux) to completely deny the `module_request` permission to containers, preventing the kernel from loading modules for containers under any circumstances. (Pods would still be able to use modules that had been loaded manually, or modules that were loaded by the kernel on behalf of some more-privileged process.)*

## **Restricting network access**

*The [network policies](#) for a namespace allows application authors to restrict which pods in other namespaces may access pods and ports within their namespaces. Many of the supported [Kubernetes networking providers](#) now respect network policy.*

*Quota and limit ranges can also be used to control whether users may request node ports or load balanced services, which on many clusters can control whether those users applications are visible outside of the cluster.*

*Additional protections may be available that control network rules on a per plugin or per environment basis, such as per-node firewalls, physically separating cluster nodes to prevent cross talk, or advanced networking policy.*

## **Restricting cloud metadata API access**

*Cloud platforms (AWS, Azure, GCE, etc.) often expose metadata services locally to instances. By default these APIs are accessible by pods running on an instance and can contain cloud credentials for that node, or provisioning data such as kubelet credentials. These credentials can be used to escalate within the cluster or to other cloud services under the same account.*

*When running Kubernetes on a cloud platform limit permissions given to instance credentials, use [network policies](#) to restrict pod access to the metadata API, and avoid using provisioning data to deliver secrets.*

## **Controlling which nodes pods may access**

*By default, there are no restrictions on which nodes may run a pod. Kubernetes offers a [rich set of policies for controlling placement of pods](#)*

[onto nodes](#) and the [taint based pod placement and eviction](#) that are available to end users. For many clusters use of these policies to separate workloads can be a convention that authors adopt or enforce via tooling.

As an administrator, a beta admission plugin `PodNodeSelector` can be used to force pods within a namespace to default or require a specific node selector, and if end users cannot alter namespaces, this can strongly limit the placement of all of the pods in a specific workload.

## **Protecting cluster components from compromise**

This section describes some common patterns for protecting clusters from compromise.

### **Restrict access to etcd**

Write access to the etcd backend for the API is equivalent to gaining root on the entire cluster, and read access can be used to escalate fairly quickly. Administrators should always use strong credentials from the API servers to their etcd server, such as mutual auth via TLS client certificates, and it is often recommended to isolate the etcd servers behind a firewall that only the API servers may access.

**Caution:** Allowing other components within the cluster to access the master etcd instance with read or write access to the full keyspace is equivalent to granting cluster-admin access. Using separate etcd instances for non-master components or using etcd ACLs to restrict read and write access to a subset of the keyspace is strongly recommended.

### **Enable audit logging**

The [audit logger](#) is a beta feature that records actions taken by the API for later analysis in the event of a compromise. It is recommended to enable audit logging and archive the audit file on a secure server.

### **Restrict access to alpha or beta features**

Alpha and beta Kubernetes features are in active development and may have limitations or bugs that result in security vulnerabilities. Always assess the value an alpha or beta feature may provide against the possible risk to your security posture. When in doubt, disable features you do not use.

### **Rotate infrastructure credentials frequently**

The shorter the lifetime of a secret or credential the harder it is for an attacker to make use of that credential. Set short lifetimes on certificates and automate their rotation. Use an authentication provider that can control how long issued tokens are available and use short lifetimes where possible.

*If you use service account tokens in external integrations, plan to rotate those tokens frequently. For example, once the bootstrap phase is complete, a bootstrap token used for setting up nodes should be revoked or its authorization removed.*

## **Review third party integrations before enabling them**

*Many third party integrations to Kubernetes may alter the security profile of your cluster. When enabling an integration, always review the permissions that an extension requests before granting it access. For example, many security integrations may request access to view all secrets on your cluster which is effectively making that component a cluster admin. When in doubt, restrict the integration to functioning in a single namespace if possible.*

*Components that create pods may also be unexpectedly powerful if they can do so inside namespaces like the `kube-system` namespace, because those pods can gain access to service account secrets or run with elevated permissions if those service accounts are granted access to permissive [pod security policies](#).*

## **Encrypt secrets at rest**

*In general, the etcd database will contain any information accessible via the Kubernetes API and may grant an attacker significant visibility into the state of your cluster. Always encrypt your backups using a well reviewed backup and encryption solution, and consider using full disk encryption where possible.*

*Kubernetes supports [encryption at rest](#), a feature introduced in 1.7, and beta since 1.13. This will encrypt Secret resources in etcd, preventing parties that gain access to your etcd backups from viewing the content of those secrets. While this feature is currently beta, it offers an additional level of defense when backups are not encrypted or an attacker gains read access to etcd.*

## **Receiving alerts for security updates and reporting vulnerabilities**

*Join the [kubernetes-announce](#) group for emails about security announcements. See the [security reporting](#) page for more on how to report vulnerabilities.*

## **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](#).*

Last modified August 07, 2020 at 8:40 PM PST: [Tune links in tasks section](#)  
[\(2/2\) \(92ae1a9cf\)](#)

[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Controlling access to the Kubernetes API](#)
  - [Use Transport Layer Security \(TLS\) for all API traffic](#)
  - [API Authentication](#)
  - [API Authorization](#)
- [Controlling access to the Kubelet](#)
- [Controlling the capabilities of a workload or user at runtime](#)
  - [Limiting resource usage on a cluster](#)
  - [Controlling what privileges containers run with](#)
  - [Preventing containers from loading unwanted kernel modules](#)
  - [Restricting network access](#)
  - [Restricting cloud metadata API access](#)
  - [Controlling which nodes pods may access](#)
- [Protecting cluster components from compromise](#)
  - [Restrict access to etcd](#)
  - [Enable audit logging](#)
  - [Restrict access to alpha or beta features](#)
  - [Rotate infrastructure credentials frequently](#)
  - [Review third party integrations before enabling them](#)
  - [Encrypt secrets at rest](#)
  - [Receiving alerts for security updates and reporting vulnerabilities](#)

## **Set Kubelet parameters via a config file**

**FEATURE STATE:** Kubernetes v1.10 [beta]

A subset of the Kubelet's configuration parameters may be set via an on-disk config file, as a substitute for command-line flags. This functionality is considered beta in v1.10.

Providing parameters via a config file is the recommended approach because it simplifies node deployment and configuration management.

### **Before you begin**

- A v1.10 or higher Kubelet binary must be installed for beta functionality.

### **Create the config file**

The subset of the Kubelet's configuration that can be configured via a file is defined by the KubeletConfiguration struct [here \(v1beta1\)](#).

*The configuration file must be a JSON or YAML representation of the parameters in this struct. Make sure the Kubelet has read permissions on the file.*

*Here is an example of what this file might look like:*

```
apiVersion: kubelet.config.k8s.io/v1beta1
kind: KubeletConfiguration
evictionHard:
    memory.available: "200Mi"
```

*In the example, the Kubelet is configured to evict Pods when available memory drops below 200Mi. All other Kubelet configuration values are left at their built-in defaults, unless overridden by flags. Command line flags which target the same value as a config file will override that value.*

*For a trick to generate a configuration file from a live node, see [Reconfigure a Node's Kubelet in a Live Cluster](#).*

## **Start a Kubelet process configured via the config file**

*Start the Kubelet with the --config flag set to the path of the Kubelet's config file. The Kubelet will then load its config from this file.*

*Note that command line flags which target the same value as a config file will override that value. This helps ensure backwards compatibility with the command-line API.*

*Note that relative file paths in the Kubelet config file are resolved relative to the location of the Kubelet config file, whereas relative paths in command line flags are resolved relative to the Kubelet's current working directory.*

*Note that some default values differ between command-line flags and the Kubelet config file. If --config is provided and the values are not specified via the command line, the defaults for the KubeletConfiguration version apply. In the above example, this version is kubelet.config.k8s.io/v1beta1.*

## **Relationship to Dynamic Kubelet Config**

*If you are using the [Dynamic Kubelet Configuration](#) feature, the combination of configuration provided via --config and any flags which override these values is considered the default "last known good" configuration by the automatic rollback mechanism.*

## **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](#).

Last modified May 30, 2020 at 3:10 PM PST: [add en pages \(ecc27bbbe\)](#)  
[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Create the config file](#)
- [Start a Kubelet process configured via the config file](#)
- [Relationship to Dynamic Kubelet Config](#)

# Set up High-Availability Kubernetes Masters

**FEATURE STATE:** Kubernetes v1.5 [alpha]

You can replicate Kubernetes masters in `kube-up` or `kube-down` scripts for Google Compute Engine. This document describes how to use `kube-up/down` scripts to manage highly available (HA) masters and how HA masters are implemented for use with GCE.

## Before you begin

You need to have a Kubernetes cluster, and the `kubectl` command-line tool must be configured to communicate with your cluster. If you do not already have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:

- [Katacoda](#)
- [Play with Kubernetes](#)

To check the version, enter `kubectl version`.

## Starting an HA-compatible cluster

To create a new HA-compatible cluster, you must set the following flags in your `kube-up` script:

- `MULTIZONE=true` - to prevent removal of master replicas kubelets from zones different than server's default zone. Required if you want to run master replicas in different zones, which is recommended.
- `ENABLE_ETCD_QUORUM_READ=true` - to ensure that reads from all API servers will return most up-to-date data. If true, reads will be directed

*to leader etcd replica. Setting this value to true is optional: reads will be more reliable but will also be slower.*

*Optionally, you can specify a GCE zone where the first master replica is to be created. Set the following flag:*

- `KUBE_GCE_ZONE=zone` - zone where the first master replica will run.

*The following sample command sets up a HA-compatible cluster in the GCE zone europe-west1-b:*

```
MULTIZONE=true KUBE_GCE_ZONE=europe-west1-b ENABLE_ETCD_QUORUM_READS=true ./cluster/kube-up.sh
```

*Note that the commands above create a cluster with one master; however, you can add new master replicas to the cluster with subsequent commands.*

## **Adding a new master replica**

*After you have created an HA-compatible cluster, you can add master replicas to it. You add master replicas by using a `kube-up` script with the following flags:*

- `KUBE_REPLICATE_EXISTING_MASTER=true` - to create a replica of an existing master.
- `KUBE_GCE_ZONE=zone` - zone where the master replica will run. Must be in the same region as other replicas' zones.

*You don't need to set the `MULTIZONE` or `ENABLE_ETCD_QUORUM_READS` flags, as those are inherited from when you started your HA-compatible cluster.*

*The following sample command replicates the master on an existing HA-compatible cluster:*

```
KUBE_GCE_ZONE=europe-west1-c KUBE_REPLICATE_EXISTING_MASTER=true ./cluster/kube-up.sh
```

## **Removing a master replica**

You can remove a master replica from an HA cluster by using a `kube-down` script with the following flags:

- `KUBE_DELETE_NODES=false` - to restrain deletion of kubelets.
- `KUBE_GCE_ZONE=zone` - the zone from where master replica will be removed.
- `KUBE_REPLICA_NAME=replica_name` - (optional) the name of master replica to remove. If empty: any replica from the given zone will be removed.

The following sample command removes a master replica from an existing HA cluster:

```
KUBE_DELETE_NODES=false KUBE_GCE_ZONE=europe-west1-c ./cluster/kube-down.sh
```

## **Handling master replica failures**

If one of the master replicas in your HA cluster fails, the best practice is to remove the replica from your cluster and add a new replica in the same zone. The following sample commands demonstrate this process:

1. Remove the broken replica:

```
KUBE_DELETE_NODES=false KUBE_GCE_ZONE=replica_zone KUBE_REPLICA_NAME=replica_name ./cluster/kube-down.sh
```

1. Add a new replica in place of the old one:

```
KUBE_GCE_ZONE=replica-zone KUBE_REPLICATE_EXISTING_MASTER=true ./cluster/kube-up.sh
```

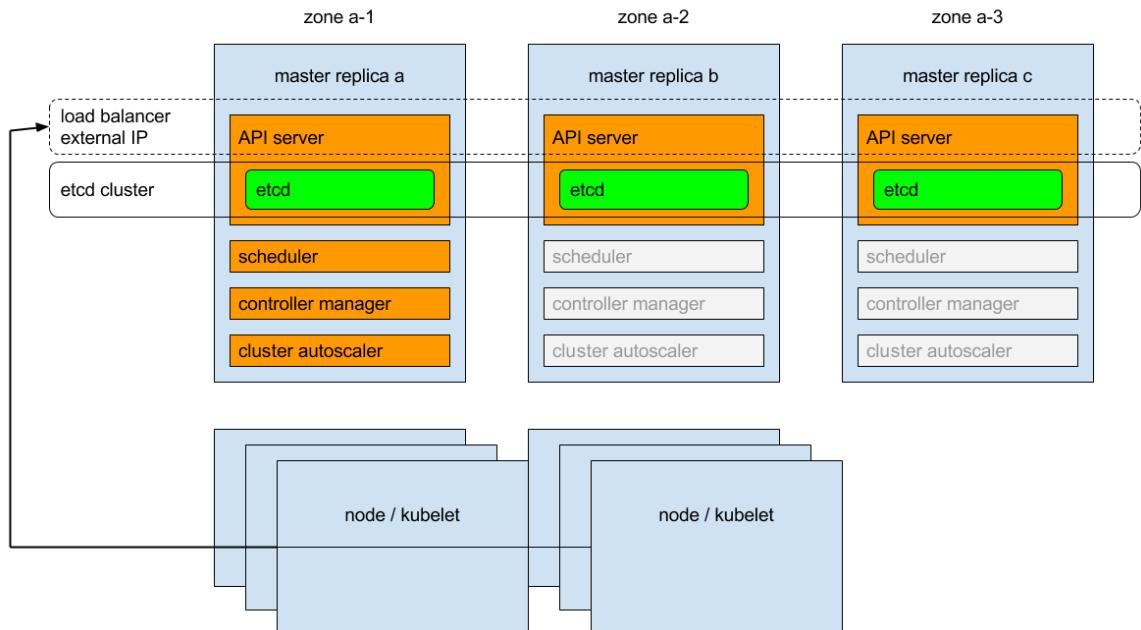
## **Best practices for replicating masters for HA clusters**

- Try to place master replicas in different zones. During a zone failure, all masters placed inside the zone will fail. To survive zone failure, also place nodes in multiple zones (see [multiple-zones](#) for details).

*Do not use a cluster with two master replicas. Consensus on a two-replica cluster requires both replicas running when changing persistent state. As a result, both replicas are needed and a failure of any replica turns cluster into majority failure state. A two-replica cluster is thus inferior, in terms of HA, to a single replica cluster.*

- When you add a master replica, cluster state (etcd) is copied to a new instance. If the cluster is large, it may take a long time to duplicate its state. This operation may be sped up by migrating etcd data directory, as described [here](#) (we are considering adding support for etcd data dir migration in future).

## Implementation notes



## Overview

*Each of master replicas will run the following components in the following mode:*

- *etcd instance: all instances will be clustered together using consensus;*

- *API server: each server will talk to local etcd - all API servers in the cluster will be available;*
- *controllers, scheduler, and cluster auto-scaler: will use lease mechanism - only one instance of each of them will be active in the cluster;*
- *add-on manager: each manager will work independently trying to keep add-ons in sync.*

*In addition, there will be a load balancer in front of API servers that will route external and internal traffic to them.*

## **Load balancing**

*When starting the second master replica, a load balancer containing the two replicas will be created and the IP address of the first replica will be promoted to IP address of load balancer. Similarly, after removal of the penultimate master replica, the load balancer will be removed and its IP address will be assigned to the last remaining replica. Please note that creation and removal of load balancer are complex operations and it may take some time (~20 minutes) for them to propagate.*

## **Master service & kubeblets**

*Instead of trying to keep an up-to-date list of Kubernetes apiserver in the Kubernetes service, the system directs all traffic to the external IP:*

- *in one master cluster the IP points to the single master,*
- *in multi-master cluster the IP points to the load balancer in-front of the masters.*

*Similarly, the external IP will be used by kubeblets to communicate with master.*

## **Master certificates**

*Kubernetes generates Master TLS certificates for the external public IP and local IP for each replica. There are no certificates for the ephemeral public IP for replicas; to access a replica via its ephemeral public IP, you must skip TLS verification.*

## **Clustering etcd**

To allow etcd clustering, ports needed to communicate between etcd instances will be opened (for inside cluster communication). To make such deployment secure, communication between etcd instances is authorized using SSL.

## **API server identity**

**FEATURE STATE:** Kubernetes v1.20 [alpha]

The API Server Identity feature is controlled by a [feature gate](#) and is not enabled by default. You can activate API Server Identity by enabling the feature gate named `APIServerIdentity` when you start the [API Server](#):

```
kube-apiserver \
--feature-gates=APIServerIdentity=true \
# and other flags as usual
```

During bootstrap, each kube-apiserver assigns a unique ID to itself. The ID is in the format of `kube-apiserver-{UUID}`. Each kube-apiserver creates a [Lease](#) in the `kube-system` [namespaces](#). The Lease name is the unique ID for the kube-apiserver. The Lease contains a label `k8s.io/component=kube-apiserver`. Each kube-apiserver refreshes its Lease every `IdentityLeaseRenewIntervalSeconds` (defaults to 10s). Each kube-apiserver also checks all the kube-apiserver identity Leases every `IdentityLeaseDurationSeconds` (defaults to 3600s), and deletes Leases that hasn't got refreshed for more than `IdentityLeaseDurationSeconds`. `IdentityLeaseRenewIntervalSeconds` and `IdentityLeaseDurationSeconds` can be configured by kube-apiserver flags `identity-lease-renew-interval-seconds` and `identity-lease-duration-seconds`.

Enabling this feature is a prerequisite for using features that involve HA API server coordination (for example, the `StorageVersionAPI` feature gate).

## **Additional reading**

[Automated HA master deployment - design doc](#)

## **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](#).

Last modified November 19, 2020 at 4:44 PM PST: [document kube-apiserver identity \(2ad9e0239\)](#)

[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Starting an HA-compatible cluster](#)
- [Adding a new master replica](#)
- [Removing a master replica](#)
- [Handling master replica failures](#)
- [Best practices for replicating masters for HA clusters](#)
- [Implementation notes](#)
  - [Overview](#)
  - [Load balancing](#)
  - [Master service & kubelets](#)
  - [Master certificates](#)
  - [Clustering etcd](#)
  - [API server identity](#)
- [Additional reading](#)

# Share a Cluster with Namespaces

This page shows how to view, work in, and delete [namespaces](#). The page also shows how to use Kubernetes namespaces to subdivide your cluster.

## Before you begin

- Have an [existing Kubernetes cluster](#).
- You have a basic understanding of Kubernetes [Pods](#), [Services](#), and [Deployments](#).

## Viewing namespaces

1. List the current namespaces in a cluster using:

```
kubectl get namespaces
```

NAME	STATUS	AGE
default	Active	11d
kube-system	Active	11d
kube-public	Active	11d

*Kubernetes starts with three initial namespaces:*

- *default* The default namespace for objects with no other namespace
- *kube-system* The namespace for objects created by the Kubernetes system
- *kube-public* This namespace is created automatically and is readable by all users (including those not authenticated). This namespace is mostly reserved for cluster usage, in case that some resources should be visible and readable publicly throughout the whole cluster. The public aspect of this namespace is only a convention, not a requirement.

*You can also get the summary of a specific namespace using:*

```
kubectl get namespaces <name>
```

*Or you can get detailed information with:*

```
kubectl describe namespaces <name>
```

```
Name:           default
Labels:         <none>
Annotations:   <none>
Status:        Active
```

*No resource quota.*

```
Resource Limits
 Type      Resource     Min Max Default
 ----      -----      -  -    -
 Container   cpu          -   -    100m
```

*Note that these details show both resource quota (if present) as well as resource limit ranges.*

*Resource quota tracks aggregate usage of resources in the Namespace and allows cluster operators to define Hard resource usage limits that a Namespace may consume.*

*A limit range defines min/max constraints on the amount of resources a single entity can consume in a Namespace.*

*See [Admission control: Limit Range](#)*

*A namespace can be in one of two phases:*

- *Active* the namespace is in use
- *Terminating* the namespace is being deleted, and can not be used for new objects

*See the [design doc](#) for more details.*

## **Creating a new namespace**

**Note:** Avoid creating namespace with prefix `kube-`, since it is reserved for Kubernetes system namespaces.

1. Create a new YAML file called `my-namespace.yaml` with the contents:

```
apiVersion: v1
kind: Namespace
metadata:
  name: <insert-namespace-name-here>
```

Then run:

```
kubectl create -f ./my-namespace.yaml
```

2. Alternatively, you can create namespace using below command:

```
kubectl create namespace <insert-namespace-name-here>
```

The name of your namespace must be a valid [DNS label](#).

There's an optional field `finalizers`, which allows observables to purge resources whenever the namespace is deleted. Keep in mind that if you specify a nonexistent finalizer, the namespace will be created but will get stuck in the `Terminating` state if the user tries to delete it.

More information on `finalizers` can be found in the namespace [design doc](#).

## **Deleting a namespace**

Delete a namespace with

```
kubectl delete namespaces <insert-some-namespace-name>
```

**Warning:** This deletes everything under the namespace!

This delete is asynchronous, so for a time you will see the namespace in the `Terminating` state.

## **Subdividing your cluster using Kubernetes namespaces**

1. Understand the default namespace

By default, a Kubernetes cluster will instantiate a default namespace when provisioning the cluster to hold the default set of Pods, Services, and Deployments used by the cluster.

Assuming you have a fresh cluster, you can introspect the available namespaces by doing the following:

```
kubectl get namespaces
```

NAME	STATUS	AGE
default	Active	13m

## 2. Create new namespaces

*For this exercise, we will create two additional Kubernetes namespaces to hold our content.*

*In a scenario where an organization is using a shared Kubernetes cluster for development and production use cases:*

*The development team would like to maintain a space in the cluster where they can get a view on the list of Pods, Services, and Deployments they use to build and run their application. In this space, Kubernetes resources come and go, and the restrictions on who can or cannot modify resources are relaxed to enable agile development.*

*The operations team would like to maintain a space in the cluster where they can enforce strict procedures on who can or cannot manipulate the set of Pods, Services, and Deployments that run the production site.*

*One pattern this organization could follow is to partition the Kubernetes cluster into two namespaces: development and production.*

*Let's create two new namespaces to hold our work.*

*Create the development namespace using kubectl:*

```
kubectl create -f https://k8s.io/examples/admin/namespace-dev.json
```

*And then let's create the production namespace using kubectl:*

```
kubectl create -f https://k8s.io/examples/admin/namespace-prod.json
```

*To be sure things are right, list all of the namespaces in our cluster.*

```
kubectl get namespaces --show-labels
```

NAME	STATUS	AGE	LABELS
default	Active	32m	<none>
development	Active	29s	name=development
production	Active	23s	name=production

## 3. Create pods in each namespace

*A Kubernetes namespace provides the scope for Pods, Services, and Deployments in the cluster.*

*Users interacting with one namespace do not see the content in another namespace.*

*To demonstrate this, let's spin up a simple Deployment and Pods in the development namespace.*

```
kubectl create deployment snowflake --image=k8s.gcr.io/  
serve_hostname -n=development --replicas=2
```

*We have just created a deployment whose replica size is 2 that is running the pod called snowflake with a basic container that just serves the hostname.*

```
kubectl get deployment -n=development
```

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
snowflake	2/2	2	2	2m

```
kubectl get pods -l app=snowflake -n=development
```

NAME	READY	STATUS	RESTARTS
snowflake-3968820950-9dgr8	1/1	Running	0
snowflake-3968820950-vgc4n	1/1	Running	0

*And this is great, developers are able to do what they want, and they do not have to worry about affecting content in the production namespace.*

*Let's switch to the production namespace and show how resources in one namespace are hidden from the other.*

*The production namespace should be empty, and the following commands should return nothing.*

```
kubectl get deployment -n=production  
kubectl get pods -n=production
```

*Production likes to run cattle, so let's create some cattle pods.*

```
kubectl create deployment cattle --image=k8s.gcr.io/  
serve_hostname -n=production  
kubectl scale deployment cattle --replicas=5 -n=production
```

```
kubectl get deployment -n=production
```

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
cattle	5/5	5	5	10s

```
kubectl get pods -l app=cattle -n=production
```

NAME	READY	STATUS	RESTARTS	AGE
cattle-2263376956-41xy6	1/1	Running	0	34s
cattle-2263376956-kw466	1/1	Running	0	34s
cattle-2263376956-n4v97	1/1	Running	0	34s
cattle-2263376956-p5p3i	1/1	Running	0	34s
cattle-2263376956-sxpth	1/1	Running	0	34s

*At this point, it should be clear that the resources users create in one namespace are hidden from the other namespace.*

*As the policy support in Kubernetes evolves, we will extend this scenario to show how you can provide different authorization rules for each namespace.*

## ***Understanding the motivation for using namespaces***

*A single cluster should be able to satisfy the needs of multiple users or groups of users (henceforth a 'user community').*

*Kubernetes namespaces help different projects, teams, or customers to share a Kubernetes cluster.*

*It does this by providing the following:*

1. A scope for [Names](#).
2. A mechanism to attach authorization and policy to a subsection of the cluster.

*Use of multiple namespaces is optional.*

*Each user community wants to be able to work in isolation from other communities.*

*Each user community has its own:*

1. resources (pods, services, replication controllers, etc.)
2. policies (who can or cannot perform actions in their community)
3. constraints (this community is allowed this much quota, etc.)

*A cluster operator may create a Namespace for each unique user community.*

*The Namespace provides a unique scope for:*

1. named resources (to avoid basic naming collisions)
2. delegated management authority to trusted users
3. ability to limit community resource consumption

*Use cases include:*

1. As a cluster operator, I want to support multiple user communities on a single cluster.

2. As a cluster operator, I want to delegate authority to partitions of the cluster to trusted users in those communities.
3. As a cluster operator, I want to limit the amount of resources each community can consume in order to limit the impact to other communities using the cluster.
4. As a cluster user, I want to interact with resources that are pertinent to my user community in isolation of what other user communities are doing on the cluster.

## **Understanding namespaces and DNS**

When you create a [Service](#), it creates a corresponding [DNS entry](#). This entry is of the form <service-name>. <namespace-name>. svc. cluster. local, which means that if a container just uses <service-name> it will resolve to the service which is local to a namespace. This is useful for using the same configuration across multiple namespaces such as Development, Staging and Production. If you want to reach across namespaces, you need to use the fully qualified domain name (FQDN).

## **What's next**

- Learn more about [setting the namespace preference](#).
- Learn more about [setting the namespace for a request](#)
- See [namespaces design](#).

## **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](#).

Last modified December 17, 2020 at 4:02 PM PST: [Update namespaces.md \(fce54f2c\)](#)

[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Viewing namespaces](#)
- [Creating a new namespace](#)
- [Deleting a namespace](#)
- [Subdividing your cluster using Kubernetes namespaces](#)
- [Understanding the motivation for using namespaces](#)
- [Understanding namespaces and DNS](#)
- [What's next](#)

# Upgrade A Cluster

This page provides an overview of the steps you should follow to upgrade a Kubernetes cluster.

The way that you upgrade a cluster depends on how you initially deployed it and on any subsequent changes.

At a high level, the steps you perform are:

- Upgrade the [control plane](#)
- Upgrade the nodes in your cluster
- Upgrade clients such as [kubectl](#)
- Adjust manifests and other resources based on the API changes that accompany the new Kubernetes version

## Before you begin

You must have an existing cluster. This page is about upgrading from Kubernetes 1.19 to Kubernetes 1.20. If your cluster is not currently running Kubernetes 1.19 then please check the documentation for the version of Kubernetes that you plan to upgrade to.

## Upgrade approaches

### kubeadm

If your cluster was deployed using the `kubeadm` tool, refer to [Upgrading kubeadm clusters](#) for detailed information on how to upgrade the cluster.

Once you have upgraded the cluster, remember to [install the latest version of kubectl](#).

### Manual deployments

**Caution:** These steps do not account for third-party extensions such as network and storage plugins.

You should manually update the control plane following this sequence:

- `etcd` (all instances)
- `kube-apiserver` (all control plane hosts)
- `kube-controller-manager`
- `kube-scheduler`
- `cloud controller manager`, if you use one

At this point you should [install the latest version of kubectl](#).

For each node in your cluster, [drain](#) that node and then either replace it with a new node that uses the 1.20 kubelet, or upgrade the kubelet on that node and bring the node back into service.

## Other deployments

Refer to the documentation for your cluster deployment tool to learn the recommended set up steps for maintenance.

## Post-upgrade tasks

### Switch your cluster's storage API version

The objects that are serialized into etcd for a cluster's internal representation of the Kubernetes resources active in the cluster are written using a particular version of the API.

When the supported API changes, these objects may need to be rewritten in the newer API. Failure to do this will eventually result in resources that are no longer decodable or usable by the Kubernetes API server.

For each affected object, fetch it using the latest supported API and then write it back also using the latest supported API.

### Update manifests

Upgrading to a new Kubernetes version can provide new APIs.

You can use `kubectl convert` command to convert manifests between different API versions. For example:

```
kubectl convert -f pod.yaml --output-version v1
```

The `kubectl` tool replaces the contents of `pod.yaml` with a manifest that sets `kind` to `Pod` (unchanged), but with a revised `apiVersion`.

## 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](#).

Last modified October 07, 2020 at 7:16 PM PST: [Revise cluster management task \(59dcd57cc\)](#)

[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)

- [Upgrade approaches](#)
  - [kubeadm](#)
  - [Manual deployments](#)
  - [Other deployments](#)
- [Post-upgrade tasks](#)
  - [Switch your cluster's storage API version](#)
  - [Update manifests](#)

# **Using a KMS provider for data encryption**

*This page shows how to configure a Key Management Service (KMS) provider and plugin to enable secret data encryption.*

## **Before you begin**

- You need to have a Kubernetes cluster, and the `kubectl` command-line tool must be configured to communicate with your cluster. If you do not already have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:
    - [Katacoda](#)
    - [Play with Kubernetes](#)
- To check the version, enter `kubectl version`.*
- Kubernetes version 1.10.0 or later is required
  - etcd v3 or later is required

**FEATURE STATE:** Kubernetes v1.12 [beta]

*The KMS encryption provider uses an envelope encryption scheme to encrypt data in etcd. The data is encrypted using a data encryption key (DEK); a new DEK is generated for each encryption. The DEKs are encrypted with a key encryption key (KEK) that is stored and managed in a remote KMS. The KMS provider uses gRPC to communicate with a specific KMS plugin. The KMS plugin, which is implemented as a gRPC server and deployed on the same host(s) as the Kubernetes master(s), is responsible for all communication with the remote KMS.*

## **Configuring the KMS provider**

*To configure a KMS provider on the API server, include a provider of type `km s` in the providers array in the encryption configuration file and set the following properties:*

- `name`: Display name of the KMS plugin.
- `endpoint`: Listen address of the gRPC server (KMS plugin). The endpoint is a UNIX domain socket.

- *cachesize*: Number of data encryption keys (DEKs) to be cached in the clear. When cached, DEKs can be used without another call to the KMS; whereas DEKs that are not cached require a call to the KMS to unwrap.
- *timeout*: How long should kube-apiserver wait for `kms-plugin` to respond before returning an error (default is 3 seconds).

See [Understanding the encryption at rest configuration](#).

## **Implementing a KMS plugin**

To implement a KMS plugin, you can develop a new plugin gRPC server or enable a KMS plugin already provided by your cloud provider. You then integrate the plugin with the remote KMS and deploy it on the Kubernetes master.

### **Enabling the KMS supported by your cloud provider**

Refer to your cloud provider for instructions on enabling the cloud provider-specific KMS plugin.

### **Developing a KMS plugin gRPC server**

You can develop a KMS plugin gRPC server using a stub file available for Go. For other languages, you use a proto file to create a stub file that you can use to develop the gRPC server code.

- Using Go: Use the functions and data structures in the stub file: [`service.pb.go`](#) to develop the gRPC server code
- Using languages other than Go: Use the protoc compiler with the proto file: [`service.proto`](#) to generate a stub file for the specific language

Then use the functions and data structures in the stub file to develop the server code.

#### **Notes:**

- *kms plugin version*: `v1beta1`

In response to procedure call `Version`, a compatible KMS plugin should return `v1beta1` as `VersionResponse.version`.

- *message version*: `v1beta1`

All messages from KMS provider have the `version` field set to current version `v1beta1`.

- *protocol*: UNIX domain socket (`unix`)

The gRPC server should listen at UNIX domain socket.

## **Integrating a KMS plugin with the remote KMS**

The KMS plugin can communicate with the remote KMS using any protocol supported by the KMS. All configuration data, including authentication credentials the KMS plugin uses to communicate with the remote KMS, are stored and managed by the KMS plugin independently. The KMS plugin can encode the ciphertext with additional metadata that may be required before sending it to the KMS for decryption.

### **Deploying the KMS plugin**

Ensure that the KMS plugin runs on the same host(s) as the Kubernetes master(s).

## **Encrypting your data with the KMS provider**

To encrypt the data:

1. Create a new encryption configuration file using the appropriate properties for the `kms` provider:

```
apiVersion: apiserver.config.k8s.io/v1
kind: EncryptionConfiguration
resources:
  - resources:
    - secrets
providers:
  - kms:
    name: myKmsPlugin
    endpoint: unix:///tmp/socketfile.sock
    cachesize: 100
    timeout: 3s
  - identity: {}
```

2. Set the `--encryption-provider-config` flag on the `kube-apiserver` to point to the location of the configuration file.
3. Restart your API server.

## **Verifying that the data is encrypted**

Data is encrypted when written to etcd. After restarting your `kube-apiserver`, any newly created or updated secret should be encrypted when stored. To verify, you can use the `etcdctl` command line program to retrieve the contents of your secret.

1. Create a new secret called `secret1` in the default namespace:

```
kubectl create secret generic secret1 -n default --from-literal=mykey=mydata
```

- Using the etcdctl command line, read that secret out of etcd:
- 

```
ETCDCTL_API=3 etcdctl get /kubernetes.io/secrets/default/  
secret1 [...] | hexdump -C
```

where [...] must be the additional arguments for connecting to the etcd server.

- Verify the stored secret is prefixed with `k8s:enc:kms:v1:`, which indicates that the `kms` provider has encrypted the resulting data.
- Verify that the secret is correctly decrypted when retrieved via the API:

```
kubectl describe secret secret1 -n default
```

should match `mykey: mydata`

## Ensuring all secrets are encrypted

Because secrets are encrypted on write, performing an update on a secret encrypts that content.

The following command reads all secrets and then updates them to apply server side encryption. If an error occurs due to a conflicting write, retry the command. For larger clusters, you may wish to subdivide the secrets by namespace or script an update.

```
kubectl get secrets --all-namespaces -o json | kubectl replace -  
f -
```

## Switching from a local encryption provider to the KMS provider

To switch from a local encryption provider to the `kms` provider and re-encrypt all of the secrets:

- Add the `kms` provider as the first entry in the configuration file as shown in the following example.

```
apiVersion: apiserver.config.k8s.io/v1  
kind: EncryptionConfiguration  
resources:  
- resources:  
- secrets  
providers:  
- kms:  
name : myKmsPlugin  
endpoint: unix:///tmp/socketfile.sock  
cachesize: 100  
- aescbc:  
keys:
```

```
- name: key1  
secret: <BASE 64 ENCODED SECRET>
```

2. Restart all `kube-apiserver` processes.
3. Run the following command to force all secrets to be re-encrypted using the `kms` provider.

```
kubectl get secrets --all-namespaces -o json | kubectl  
replace -f -
```

## Disabling encryption at rest

To disable encryption at rest:

1. Place the `identity` provider as the first entry in the configuration file:

```
apiVersion: apiserver.config.k8s.io/v1  
kind: EncryptionConfiguration  
resources:  
- resources:  
  - secrets  
providers:  
- identity: {}  
- kms:  
  name : myKmsPlugin  
  endpoint: unix:///tmp/socketfile.sock  
  cachesize: 100
```

2. Restart all `kube-apiserver` processes.
3. Run the following command to force all secrets to be decrypted.

```
kubectl get secrets --all-namespaces -o json | kubectl  
replace -f -
```

## 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](#).

Last modified July 04, 2020 at 12:31 PM PST: [Fix indentation of list and code snippet for kms-provider \(f03c18798\)](#)

[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Configuring the KMS provider](#)

- [Implementing a KMS plugin](#)
  - [Enabling the KMS supported by your cloud provider](#)
  - [Developing a KMS plugin gRPC server](#)
  - [Integrating a KMS plugin with the remote KMS](#)
  - [Deploying the KMS plugin](#)
- [Encrypting your data with the KMS provider](#)
- [Verifying that the data is encrypted](#)
- [Ensuring all secrets are encrypted](#)
- [Switching from a local encryption provider to the KMS provider](#)
- [Disabling encryption at rest](#)

# **Using CoreDNS for Service Discovery**

*This page describes the CoreDNS upgrade process and how to install CoreDNS instead of kube-dns.*

## **Before you begin**

*You need to have a Kubernetes cluster, and the kubectl command-line tool must be configured to communicate with your cluster. If you do not already have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:*

- [Katacoda](#)
- [Play with Kubernetes](#)

*Your Kubernetes server must be at or later than version v1.9. To check the version, enter kubectl version.*

## **About CoreDNS**

[CoreDNS](#) is a flexible, extensible DNS server that can serve as the Kubernetes cluster DNS. Like Kubernetes, the CoreDNS project is hosted by the [CNCF](#).

*You can use CoreDNS instead of kube-dns in your cluster by replacing kube-dns in an existing deployment, or by using tools like kubeadm that will deploy and upgrade the cluster for you.*

## **Installing CoreDNS**

*For manual deployment or replacement of kube-dns, see the documentation at the [CoreDNS GitHub project](#).*

# **Migrating to CoreDNS**

## **Upgrading an existing cluster with kubeadm**

*In Kubernetes version 1.10 and later, you can also move to CoreDNS when you use kubeadm to upgrade a cluster that is using kube-dns. In this case, kubeadm will generate the CoreDNS configuration ("Corefile") based upon the kube-dns ConfigMap, preserving configurations for federation, stub domains, and upstream name server.*

*If you are moving from kube-dns to CoreDNS, make sure to set the CoreDNS feature gate to true during an upgrade. For example, here is what a v1.11.0 upgrade would look like:*

```
kubeadm upgrade apply v1.11.0 --feature-gates=CoreDNS=true
```

*In Kubernetes version 1.13 and later the CoreDNS feature gate is removed and CoreDNS is used by default. Follow the guide outlined [here](#) if you want your upgraded cluster to use kube-dns.*

*In versions prior to 1.11 the Corefile will be **overwritten** by the one created during upgrade. **You should save your existing ConfigMap if you have customized it.** You may re-apply your customizations after the new ConfigMap is up and running.*

*If you are running CoreDNS in Kubernetes version 1.11 and later, during upgrade, your existing Corefile will be retained.*

## **Installing kube-dns instead of CoreDNS with kubeadm**

**Note:** In Kubernetes 1.11, CoreDNS has graduated to General Availability (GA) and is installed by default.

**Warning:** In Kubernetes 1.18, kube-dns usage with kubeadm has been deprecated and will be removed in a future version.

*To install kube-dns on versions prior to 1.13, set the CoreDNS feature gate value to false:*

```
kubeadm init --feature-gates=CoreDNS=false
```

For versions 1.13 and later, follow the guide outlined [here](#).

## Upgrading CoreDNS

CoreDNS is available in Kubernetes since v1.9. You can check the version of CoreDNS shipped with Kubernetes and the changes made to CoreDNS [here](#).

CoreDNS can be upgraded manually in case you want to only upgrade CoreDNS or use your own custom image. There is a helpful [guideline and walkthrough](#) available to ensure a smooth upgrade.

## Tuning CoreDNS

When resource utilisation is a concern, it may be useful to tune the configuration of CoreDNS. For more details, check out the [documentation on scaling CoreDNS](#).

## What's next

You can configure [CoreDNS](#) to support many more use cases than kube-dns by modifying the [Corefile](#). For more information, see the [CoreDNS site](#).

## 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](#).

Last modified May 30, 2020 at 3:10 PM PST: [add en pages \(ecc27bbbe\)](#)  
[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [About CoreDNS](#)
- [Installing CoreDNS](#)
- [Migrating to CoreDNS](#)
  - [Upgrading an existing cluster with kubeadm](#)
  - [Installing kube-dns instead of CoreDNS with kubeadm](#)
- [Upgrading CoreDNS](#)
- [Tuning CoreDNS](#)

- [What's next](#)

# Using NodeLocal DNSCache in Kubernetes clusters

**FEATURE STATE:** Kubernetes v1.18 [stable]

This page provides an overview of NodeLocal DNSCache feature in Kubernetes.

## Before you begin

You need to have a Kubernetes cluster, and the `kubectl` command-line tool must be configured to communicate with your cluster. If you do not already have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:

- [Katacoda](#)
- [Play with Kubernetes](#)

To check the version, enter `kubectl version`.

## Introduction

*NodeLocal DNSCache improves Cluster DNS performance by running a dns caching agent on cluster nodes as a DaemonSet. In today's architecture, Pods in ClusterFirst DNS mode reach out to a kube-dns serviceIP for DNS queries. This is translated to a kube-dns/CoreDNS endpoint via iptables rules added by kube-proxy. With this new architecture, Pods will reach out to the dns caching agent running on the same node, thereby avoiding iptables DNAT rules and connection tracking. The local caching agent will query kube-dns service for cache misses of cluster hostnames(cluster.local suffix by default).*

## Motivation

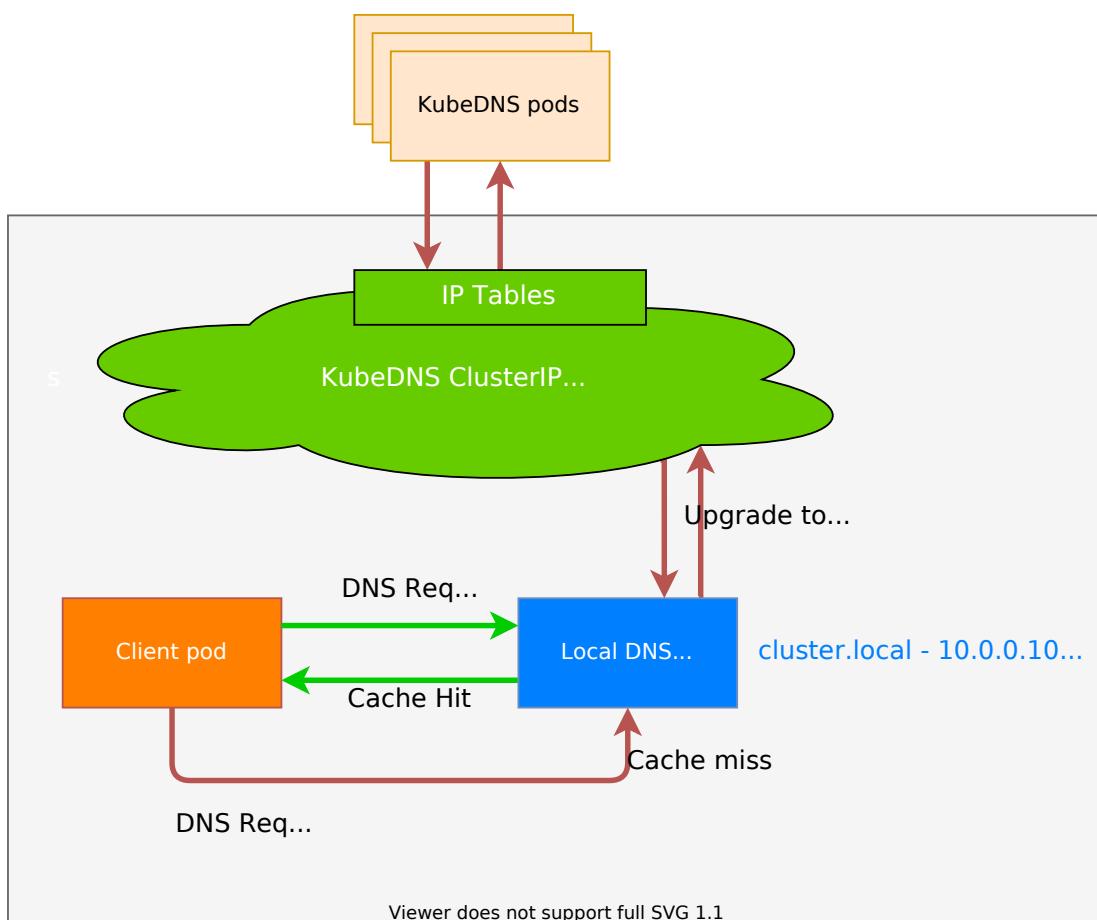
- *With the current DNS architecture, it is possible that Pods with the highest DNS QPS have to reach out to a different node, if there is no local kube-dns/CoreDNS instance. Having a local cache will help improve the latency in such scenarios.*

*Skipping iptables DNAT and connection tracking will help reduce • [conntrack races](#) and avoid UDP DNS entries filling up conntrack table.*

- *Connections from local caching agent to kube-dns service can be upgraded to TCP. TCP conntrack entries will be removed on connection close in contrast with UDP entries that have to timeout ([default nf\\_conntrack\\_udp\\_timeout](#) is 30 seconds)*
- *Upgrading DNS queries from UDP to TCP would reduce tail latency attributed to dropped UDP packets and DNS timeouts usually up to 30s (3 retries + 10s timeout). Since the nodelocal cache listens for UDP DNS queries, applications don't need to be changed.*
- *Metrics & visibility into dns requests at a node level.*
- *Negative caching can be re-enabled, thereby reducing number of queries to kube-dns service.*

## Architecture Diagram

*This is the path followed by DNS Queries after NodeLocal DNSCache is enabled:*



## ***Nodelocal DNSCache flow***

*This image shows how NodeLocal DNSCache handles DNS queries.*

## ***Configuration***

**Note:** The local listen IP address for NodeLocal DNSCache can be any IP in the 169.254.20.0/16 space or any other IP address that can be guaranteed to not collide with any existing IP. This document uses 169.254.20.10 as an example.

*This feature can be enabled using the following steps:*

- Prepare a manifest similar to the sample [`nodelocaldns.yaml`](#) and save it as `nodelocaldns.yaml`.
  - `kubedns=kubectl get svc kube-dns -n kube-system -o jsonpath={.spec.clusterIP}`
  - `domain=<cluster-domain>`
  - `localdns=<node-local-address>`

*<cluster-domain> is "cluster.local" by default. <node-local-address> is the local listen IP address chosen for NodeLocal DNSCache.*

- If kube-proxy is running in IPTABLES mode:

```
sed -i "s/_PILLAR_LOCAL_DNS_/$localdns/g; s/_PILLAR_DNS_DOMAIN_/$domain/g; s/_PILLAR_DNS_SERVER_/$kubedns/g" nodelocaldns.yaml
```

*\_PILLAR\_CLUSTER\_DNS\_ and \_PILLAR\_UPSTREAM\_SERVERS\_ will be populated by the node-local-dns pods. In this mode, node-local-dns pods listen on both the kube-dns service IP as well as <node-local-address>, so pods can lookup DNS records using either IP address.*

- If kube-proxy is running in IPVS mode:

```
sed -i "s/_PILLAR_LOCAL_DNS_/$localdns/g; s/_PILLAR_DNS_DOMAIN_/$domain/g; s/_PILLAR_DNS_SERVER_//g; s/_PILLAR_CLUSTER_DNS_/$kubedns/g" nodelocaldns.yaml
```

*In this mode, node-local-dns pods listen only on <node-local-address>. The node-local-dns interface cannot bind the kube-dns cluster IP since the interface used for IPVS loadbalancing already uses this address. \_\_PILLAR\_UPSTREAM\_SERVERS\_\_ will be populated by the node-local-dns pods.*

- Run `kubectl create -f nodelocaldns.yaml`
- If using kube-proxy in IPVS mode, --cluster-dns flag to kubelet needs to be modified to use <node-local-address> that NodeLocal DNSCache is listening on. Otherwise, there is no need to modify the value of the --cluster-dns flag, since NodeLocal DNSCache listens on both the kube-dns service IP as well as <node-local-address>.

Once enabled, node-local-dns Pods will run in the kube-system namespace on each of the cluster nodes. This Pod runs [CoreDNS](#) in cache mode, so all CoreDNS metrics exposed by the different plugins will be available on a per-node basis.

You can disable this feature by removing the DaemonSet, using `kubectl delete -f <manifest>`. You should also revert any changes you made to the kubelet configuration.

## 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](#).

Last modified May 30, 2020 at 3:10 PM PST: [add en pages \(ecc27bbbe\)](#)  
[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Introduction](#)
- [Motivation](#)
- [Architecture Diagram](#)
- [Configuration](#)

# Using sysctls in a Kubernetes Cluster

**FEATURE STATE:** Kubernetes v1.12 [beta]

This document describes how to configure and use kernel parameters within a Kubernetes cluster using the [sysctl](#) interface.

## Before you begin

You need to have a Kubernetes cluster, and the `kubectl` command-line tool must be configured to communicate with your cluster. If you do not already have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:

- [Katacoda](#)
- [Play with Kubernetes](#)

To check the version, enter `kubectl version`.

## Listing all Sysctl Parameters

In Linux, the `sysctl` interface allows an administrator to modify kernel parameters at runtime. Parameters are available via the `/proc/sys/` virtual process file system. The parameters cover various subsystems such as:

- `kernel` (common prefix: `kernel.`)
- `networking` (common prefix: `net.`)
- `virtual memory` (common prefix: `vm.`)
- `MDADM` (common prefix: `dev.`)
- More subsystems are described in [Kernel docs](#).

To get a list of all parameters, you can run

```
sudo sysctl -a
```

## Enabling Unsafe Sysctls

Sysctls are grouped into safe and unsafe sysctls. In addition to proper namespacing, a safe sysctl must be properly isolated between pods on the same node. This means that setting a safe sysctl for one pod

- must not have any influence on any other pod on the node
- must not allow to harm the node's health
- must not allow to gain CPU or memory resources outside of the resource limits of a pod.

By far, most of the namespaced sysctls are not necessarily considered safe. The following sysctls are supported in the safe set:

- `kernel.shm_rmid_forced`,
- `net.ipv4.ip_local_port_range`,
- `net.ipv4.tcp_syncookies`,
- `net.ipv4.ping_group_range` (since Kubernetes 1.18).

**Note:** The example `net.ipv4.tcp_syncookies` is not namespaced on Linux kernel version 4.4 or lower.

This list will be extended in future Kubernetes versions when the kubelet supports better isolation mechanisms.

All safe sysctls are enabled by default.

All unsafe sysctls are disabled by default and must be allowed manually by the cluster admin on a per-node basis. Pods with disabled unsafe sysctls will be scheduled, but will fail to launch.

With the warning above in mind, the cluster admin can allow certain unsafe sysctls for very special situations such as high-performance or real-time application tuning. Unsafe sysctls are enabled on a node-by-node basis with a flag of the kubelet; for example:

```
kubelet --allowed-unsafe-sysctls \
'kernel.msg*,net.core.somaxconn' ...
```

For [Minikube](#), this can be done via the extra-config flag:

```
minikube start --extra-config="kubelet.allowed-unsafe-
sysctls=kernel.msg*,net.core.somaxconn"...
```

Only namespaced sysctls can be enabled this way.

## Setting Sysctls for a Pod

A number of sysctls are namespaced in today's Linux kernels. This means that they can be set independently for each pod on a node. Only namespaced sysctls are configurable via the pod securityContext within Kubernetes.

The following sysctls are known to be namespaced. This list could change in future versions of the Linux kernel.

- `kernel.shm*`,
- `kernel.msg*`,
- `kernel.sem`,
- `fs.mqueue.*`,
- The parameters under `net.*` that can be set in container networking namespace. However, there are exceptions (e.g., `net.netfilter.nf_conntrack_max` and `net.netfilter.nf_conntrack_expect_max` can be set in container networking namespace but they are unnamespaced).

*Sysctls with no namespace are called node-level sysctls. If you need to set them, you must manually configure them on each node's operating system, or by using a DaemonSet with privileged containers.*

*Use the pod securityContext to configure namespaced sysctls. The securityContext applies to all containers in the same pod.*

*This example uses the pod securityContext to set a safe sysctl `kernel.shm_rmid_forced` and two unsafe sysctls `net.core.somaxconn` and `kernel.msgmax`. There is no distinction between safe and unsafe sysctls in the specification.*

**Warning:** Only modify sysctl parameters after you understand their effects, to avoid destabilizing your operating system.

```
apiVersion: v1
kind: Pod
metadata:
  name: sysctl-example
spec:
  securityContext:
    sysctls:
      - name: kernel.shm_rmid_forced
        value: "0"
      - name: net.core.somaxconn
        value: "1024"
      - name: kernel.msgmax
        value: "65536"
      ...

```

**Warning:** Due to their nature of being unsafe, the use of unsafe sysctls is at-your-own-risk and can lead to severe problems like wrong behavior of containers, resource shortage or complete breakage of a node.

*It is good practice to consider nodes with special sysctl settings as tainted within a cluster, and only schedule pods onto them which need those sysctl*

*settings. It is suggested to use the Kubernetes [taints and toleration feature](#) to implement this.*

*A pod with the unsafe sysctls will fail to launch on any node which has not enabled those two unsafe sysctls explicitly. As with node-level sysctls it is recommended to use [taints and toleration feature](#) or [taints on nodes](#) to schedule those pods onto the right nodes.*

## **PodSecurityPolicy**

*You can further control which sysctls can be set in pods by specifying lists of sysctls or sysctl patterns in the `forbiddenSysctls` and/or `allowedUnsafeSysctls` fields of the PodSecurityPolicy. A sysctl pattern ends with a \* character, such as `kernel.*`. A \* character on its own matches all sysctls.*

*By default, all safe sysctls are allowed.*

*Both `forbiddenSysctls` and `allowedUnsafeSysctls` are lists of plain sysctl names or sysctl patterns (which end with \*). The string \* matches all sysctls.*

*The `forbiddenSysctls` field excludes specific sysctls. You can forbid a combination of safe and unsafe sysctls in the list. To forbid setting any sysctls, use \* on its own.*

*If you specify any unsafe sysctl in the `allowedUnsafeSysctls` field and it is not present in the `forbiddenSysctls` field, that sysctl can be used in Pods using this PodSecurityPolicy. To allow all unsafe sysctls in the PodSecurityPolicy to be set, use \* on its own.*

*Do not configure these two fields such that there is overlap, meaning that a given sysctl is both allowed and forbidden.*

**Warning:** If you allow unsafe sysctls via the `allowedUnsafeSysctls` field in a PodSecurityPolicy, any pod using such a sysctl will fail to start if the sysctl is not allowed via the `--allowed-unsafe-sysctls` kubelet flag as well on that node.

*This example allows unsafe sysctls prefixed with `kernel.msg` to be set and disallows setting of the `kernel.shm_rmid_forced` sysctl.*

```
apiVersion: policy/v1beta1
kind: PodSecurityPolicy
metadata:
```

```
  name: sysctl-psp
spec:
  allowedUnsafeSysctls:
    - kernel.msg*
  forbiddenSysctls:
    - kernel.shm_rmid_forced
  ...
```

## 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](#).

Last modified June 17, 2020 at 5:16 PM PST: [Add newly introduced ping\\_group\\_range sysctl \(96dca646d\)](#)

[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Listing all Sysctl Parameters](#)
- [Enabling Unsafe Sysctls](#)
- [Setting Sysctls for a Pod](#)
- [PodSecurityPolicy](#)

# Configure Pods and Containers

Perform common configuration tasks for Pods and containers.

---

[Assign Memory Resources to Containers and Pods](#)

[Assign CPU Resources to Containers and Pods](#)

[Configure GMSA for Windows Pods and containers](#)

[Configure RunAsUserName for Windows pods and containers](#)

[Configure Quality of Service for Pods](#)

[Assign Extended Resources to a Container](#)

[Configure a Pod to Use a Volume for Storage](#)

[Configure a Pod to Use a PersistentVolume for Storage](#)

[Configure a Pod to Use a Projected Volume for Storage](#)

[Configure a Security Context for a Pod or Container](#)

[Configure Service Accounts for Pods](#)

[Pull an Image from a Private Registry](#)

[Configure Liveness, Readiness and Startup Probes](#)

[Assign Pods to Nodes](#)

[Assign Pods to Nodes using Node Affinity](#)

[Configure Pod Initialization](#)

[Attach Handlers to Container Lifecycle Events](#)

[Configure a Pod to Use a ConfigMap](#)

[Share Process Namespace between Containers in a Pod](#)

[Create static Pods](#)

[Translate a Docker Compose File to Kubernetes Resources](#)

# Assign Memory Resources to Containers and Pods

This page shows how to assign a memory request and a memory limit to a Container. A Container is guaranteed to have as much memory as it requests, but is not allowed to use more memory than its limit.

## Before you begin

You need to have a Kubernetes cluster, and the `kubectl` command-line tool must be configured to communicate with your cluster. If you do not already have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:

- [Katacoda](#)
- [Play with Kubernetes](#)

To check the version, enter `kubectl version`.

Each node in your cluster must have at least 300 MiB of memory.

A few of the steps on this page require you to run the [metrics-server](#) service in your cluster. If you have the metrics-server running, you can skip those steps.

If you are running Minikube, run the following command to enable the metrics-server:

```
minikube addons enable metrics-server
```

To see whether the metrics-server is running, or another provider of the resource metrics API (`metrics.k8s.io`), run the following command:

```
kubectl get apiservices
```

If the resource metrics API is available, the output includes a reference to `metrics.k8s.io`.

```
NAME
v1beta1.metrics.k8s.io
```

## Create a namespace

Create a namespace so that the resources you create in this exercise are isolated from the rest of your cluster.

```
kubectl create namespace mem-example
```

## Specify a memory request and a memory limit

To specify a memory request for a Container, include the `resources: requests` field in the Container's resource manifest. To specify a memory limit, include `resources: limits`.

In this exercise, you create a Pod that has one Container. The Container has a memory request of 100 MiB and a memory limit of 200 MiB. Here's the configuration file for the Pod:

[pods/resource/memory-request-limit.yaml](#)



```
apiVersion: v1
kind: Pod
metadata:
  name: memory-demo
  namespace: mem-example
spec:
  containers:
    - name: memory-demo-ctr
      image: polinux/stress
      resources:
        limits:
          memory: "200Mi"
        requests:
          memory: "100Mi"
      command: ["stress"]
      args: ["--vm", "1", "--vm-bytes", "150M", "--vm-hang", "1"]
```

The `args` section in the configuration file provides arguments for the Container when it starts. The `--vm-bytes`, `150M` arguments tell the Container to attempt to allocate 150 MiB of memory.

Create the Pod:

```
kubectl apply -f https://k8s.io/examples/pods/resource/memory-request-limit.yaml --namespace=mem-example
```

Verify that the Pod Container is running:

```
kubectl get pod memory-demo --namespace=mem-example
```

View detailed information about the Pod:

```
kubectl get pod memory-demo --output=yaml --namespace=mem-example
```

The output shows that the one Container in the Pod has a memory request of 100 MiB and a memory limit of 200 MiB.

```
...
resources:
  limits:
    memory: 200Mi
  requests:
    memory: 100Mi
...
...
```

Run `kubectl top` to fetch the metrics for the pod:

```
kubectl top pod memory-demo --namespace=mem-example
```

The output shows that the Pod is using about 162,900,000 bytes of memory, which is about 150 MiB. This is greater than the Pod's 100 MiB request, but within the Pod's 200 MiB limit.

NAME	CPU(cores)	MEMORY(bytes)
memory-demo	<something>	162856960

Delete your Pod:

```
kubectl delete pod memory-demo --namespace=mem-example
```

## Exceed a Container's memory limit

A Container can exceed its memory request if the Node has memory available. But a Container is not allowed to use more than its memory limit. If a Container allocates more memory than its limit, the Container becomes a candidate for termination. If the Container continues to consume memory beyond its limit, the Container is terminated. If a terminated Container can be restarted, the kubelet restarts it, as with any other type of runtime failure.

In this exercise, you create a Pod that attempts to allocate more memory than its limit. Here is the configuration file for a Pod that has one Container with a memory request of 50 MiB and a memory limit of 100 MiB:

[pods/resource/memory-request-limit-2.yaml](#)  


```
apiVersion: v1
kind: Pod
metadata:
  name: memory-demo-2
  namespace: mem-example
spec:
  containers:
    - name: memory-demo-2-ctr
      image: polinux/stress
      resources:
        requests:
          memory: "50Mi"
        limits:
```

```
    memory: "100Mi"
  command: [ "stress" ]
  args: [ "--vm", "1", "--vm-bytes", "250M", "--vm-hang", "1" ]
```

In the `args` section of the configuration file, you can see that the Container will attempt to allocate 250 MiB of memory, which is well above the 100 MiB limit.

Create the Pod:

```
kubectl apply -f https://k8s.io/examples/pods/resource/memory-request-limit-2.yaml --namespace=mem-example
```

View detailed information about the Pod:

```
kubectl get pod memory-demo-2 --namespace=mem-example
```

At this point, the Container might be running or killed. Repeat the preceding command until the Container is killed:

NAME	READY	STATUS	RESTARTS	AGE
memory-demo-2	0/1	0OMKilled	1	24s

Get a more detailed view of the Container status:

```
kubectl get pod memory-demo-2 --output=yaml --namespace=mem-example
```

The output shows that the Container was killed because it is out of memory (OOM):

```
lastState:
  terminated:
    containerID: docker://
65183c1877aaec2e8427bc95609cc52677a454b56fcb24340dbd22917c23b10f
    exitCode: 137
    finishedAt: 2017-06-20T20:52:19Z
    reason: OOMKilled
    startedAt: null
```

The Container in this exercise can be restarted, so the kubelet restarts it. Repeat this command several times to see that the Container is repeatedly killed and restarted:

```
kubectl get pod memory-demo-2 --namespace=mem-example
```

The output shows that the Container is killed, restarted, killed again, restarted again, and so on:

```
kubectl get pod memory-demo-2 --namespace=mem-example
NAME          READY   STATUS    RESTARTS   AGE
memory-demo-2  0/1     OOMKilled  1          37s
```

```
kubectl get pod memory-demo-2 --namespace=mem-example
NAME          READY   STATUS    RESTARTS   AGE
memory-demo-2  1/1     Running   2          40s
```

View detailed information about the Pod history:

```
kubectl describe pod memory-demo-2 --namespace=mem-example
```

The output shows that the Container starts and fails repeatedly:

```
... Normal Created  Created container with id
66a3a20aa7980e61be4922780bf9d24d1a1d8b7395c09861225b0eba1b1f8511
... Warning BackOff  Back-off restarting failed container
```

View detailed information about your cluster's Nodes:

```
kubectl describe nodes
```

The output includes a record of the Container being killed because of an out-of-memory condition:

```
Warning OOMKilling Memory cgroup out of memory: Kill process
4481 (stress) score 1994 or sacrifice child
```

Delete your Pod:

```
kubectl delete pod memory-demo-2 --namespace=mem-example
```

# **Specify a memory request that is too big for your Nodes**

*Memory requests and limits are associated with Containers, but it is useful to think of a Pod as having a memory request and limit. The memory request for the Pod is the sum of the memory requests for all the Containers in the Pod. Likewise, the memory limit for the Pod is the sum of the limits of all the Containers in the Pod.*

*Pod scheduling is based on requests. A Pod is scheduled to run on a Node only if the Node has enough available memory to satisfy the Pod's memory request.*

*In this exercise, you create a Pod that has a memory request so big that it exceeds the capacity of any Node in your cluster. Here is the configuration file for a Pod that has one Container with a request for 1000 GiB of memory, which likely exceeds the capacity of any Node in your cluster.*

[pods/resource/memory-request-limit-3.yaml](#)  


```
apiVersion: v1
kind: Pod
metadata:
  name: memory-demo-3
  namespace: mem-example
spec:
  containers:
    - name: memory-demo-3-ctr
      image: polinux/stress
      resources:
        limits:
          memory: "1000Gi"
        requests:
          memory: "1000Gi"
      command: ["stress"]
      args: ["--vm", "1", "--vm-bytes", "150M", "--vm-hang", "1"]
```

*Create the Pod:*

```
kubectl apply -f https://k8s.io/examples/pods/resource/memory-
request-limit-3.yaml --namespace=mem-example
```

*View the Pod status:*

```
kubectl get pod memory-demo-3 --namespace=mem-example
```

The output shows that the Pod status is PENDING. That is, the Pod is not scheduled to run on any Node, and it will remain in the PENDING state indefinitely:

```
kubectl get pod memory-demo-3 --namespace=mem-example
```

NAME	READY	STATUS	RESTARTS	AGE
memory-demo-3	0/1	Pending	0	25s

View detailed information about the Pod, including events:

```
kubectl describe pod memory-demo-3 --namespace=mem-example
```

The output shows that the Container cannot be scheduled because of insufficient memory on the Nodes:

Events:

... Reason	Message
---	---
... FailedScheduling	No nodes are available that match all of the following predicates:: Insufficient memory (3).

## Memory units

The memory resource is measured in bytes. You can express memory as a plain integer or a fixed-point integer with one of these suffixes: E, P, T, G, M, K, Ei, Pi, Ti, Gi, Mi, Ki. For example, the following represent approximately the same value:

```
128974848, 129e6, 129M , 123Mi
```

Delete your Pod:

```
kubectl delete pod memory-demo-3 --namespace=mem-example
```

## If you do not specify a memory limit

If you do not specify a memory limit for a Container, one of the following situations applies:

- *The Container has no upper bound on the amount of memory it uses. The Container could use all of the memory available on the Node where it is running which in turn could invoke the OOM Killer. Further, in case of an OOM Kill, a container with no resource limits will have a greater chance of being killed.*
- *The Container is running in a namespace that has a default memory limit, and the Container is automatically assigned the default limit. Cluster administrators can use a [LimitRange](#) to specify a default value for the memory limit.*

## **Motivation for memory requests and limits**

*By configuring memory requests and limits for the Containers that run in your cluster, you can make efficient use of the memory resources available on your cluster's Nodes. By keeping a Pod's memory request low, you give the Pod a good chance of being scheduled. By having a memory limit that is greater than the memory request, you accomplish two things:*

- *The Pod can have bursts of activity where it makes use of memory that happens to be available.*
- *The amount of memory a Pod can use during a burst is limited to some reasonable amount.*

## **Clean up**

*Delete your namespace. This deletes all the Pods that you created for this task:*

```
kubectl delete namespace mem-example
```

## **What's next**

### **For app developers**

- [Assign CPU Resources to Containers and Pods](#)
- [Configure Quality of Service for Pods](#)

## For cluster administrators

- [Configure Default Memory Requests and Limits for a Namespace](#)
- [Configure Default CPU Requests and Limits for a Namespace](#)
- [Configure Minimum and Maximum Memory Constraints for a Namespace](#)
- [Configure Minimum and Maximum CPU Constraints for a Namespace](#)
- [Configure Memory and CPU Quotas for a Namespace](#)
- [Configure a Pod Quota for a Namespace](#)
- [Configure Quotas for API 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](#).

Last modified October 17, 2020 at 3:21 PM PST: [update kubernetes-incubator references \(a8b6551c2\)](#)

[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Create a namespace](#)
- [Specify a memory request and a memory limit](#)
- [Exceed a Container's memory limit](#)
- [Specify a memory request that is too big for your Nodes](#)
- [Memory units](#)
- [If you do not specify a memory limit](#)
- [Motivation for memory requests and limits](#)
- [Clean up](#)
- [What's next](#)
  - [For app developers](#)
  - [For cluster administrators](#)

# Assign CPU Resources to Containers and Pods

This page shows how to assign a CPU request and a CPU limit to a container. Containers cannot use more CPU than the configured limit. Provided the system has CPU time free, a container is guaranteed to be allocated as much CPU as it requests.

## Before you begin

You need to have a Kubernetes cluster, and the `kubectl` command-line tool must be configured to communicate with your cluster. If you do not already have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:

- [Katacoda](#)
- [Play with Kubernetes](#)

To check the version, enter `kubectl version`.

Your cluster must have at least 1 CPU available for use to run the task examples.

A few of the steps on this page require you to run the [metrics-server](#) service in your cluster. If you have the metrics-server running, you can skip those steps.

If you are running [Minikube](#), run the following command to enable metrics-server:

```
minikube addons enable metrics-server
```

To see whether metrics-server (or another provider of the resource metrics API, `metrics.k8s.io`) is running, type the following command:

```
kubectl get apiservices
```

If the resource metrics API is available, the output will include a reference to `metrics.k8s.io`.

NAME
v1beta1.metrics.k8s.io

## Create a namespace

Create a [Namespace](#) so that the resources you create in this exercise are isolated from the rest of your cluster.

```
kubectl create namespace cpu-example
```

## Specify a CPU request and a CPU limit

To specify a CPU request for a container, include the `resources:requests` field in the Container resource manifest. To specify a CPU limit, include `resources:limits`.

In this exercise, you create a Pod that has one container. The container has a request of 0.5 CPU and a limit of 1 CPU. Here is the configuration file for the Pod:

[pods/resource/cpu-request-limit.yaml](#)  


```
apiVersion: v1
kind: Pod
metadata:
  name: cpu-demo
  namespace: cpu-example
spec:
  containers:
    - name: cpu-demo-ctr
      image: vish/stress
      resources:
        limits:
          cpu: "1"
        requests:
          cpu: "0.5"
      args:
        - -cpus
        - "2"
```

The `args` section of the configuration file provides arguments for the container when it starts. The `-cpus "2"` argument tells the Container to attempt to use 2 CPUs.

Create the Pod:

```
kubectl apply -f https://k8s.io/examples/pods/resource/cpu-request-limit.yaml --namespace(cpu-example)
```

Verify that the Pod is running:

```
kubectl get pod cpu-demo --namespace(cpu-example)
```

View detailed information about the Pod:

```
kubectl get pod cpu-demo --output=yaml --namespace(cpu-example)
```

The output shows that the one container in the Pod has a CPU request of 500 milliCPU and a CPU limit of 1 CPU.

```
resources:  
  limits:  
    cpu: "1"  
  requests:  
    cpu: 500m
```

Use `kubectl top` to fetch the metrics for the pod:

```
kubectl top pod cpu-demo --namespace(cpu-example)
```

This example output shows that the Pod is using 974 milliCPU, which is just a bit less than the limit of 1 CPU specified in the Pod configuration.

NAME	CPU(cores)	MEMORY(bytes)
cpu-demo	974m	<something>

Recall that by setting `-cpu "2"`, you configured the Container to attempt to use 2 CPUs, but the Container is only being allowed to use about 1 CPU. The container's CPU use is being throttled, because the container is attempting to use more CPU resources than its limit.

**Note:** Another possible explanation for the CPU use being below 1.0 is that the Node might not have enough CPU resources available. Recall that the prerequisites for this exercise require your cluster to have at least 1 CPU available for use. If your Container runs on a Node that has only 1 CPU, the Container cannot use more than 1 CPU regardless of the CPU limit specified for the Container.

## **CPU units**

*The CPU resource is measured in CPU units. One CPU, in Kubernetes, is equivalent to:*

- 1 AWS vCPU
- 1 GCP Core
- 1 Azure vCore
- 1 Hyperthread on a bare-metal Intel processor with Hyperthreading

*Fractional values are allowed. A Container that requests 0.5 CPU is guaranteed half as much CPU as a Container that requests 1 CPU. You can use the suffix m to mean milli. For example 100m CPU, 100 milliCPU, and 0.1 CPU are all the same. Precision finer than 1m is not allowed.*

*CPU is always requested as an absolute quantity, never as a relative quantity; 0.1 is the same amount of CPU on a single-core, dual-core, or 48-core machine.*

*Delete your Pod:*

```
kubectl delete pod cpu-demo --namespace(cpu-example)
```

## **Specify a CPU request that is too big for your Nodes**

*CPU requests and limits are associated with Containers, but it is useful to think of a Pod as having a CPU request and limit. The CPU request for a Pod is the sum of the CPU requests for all the Containers in the Pod. Likewise, the CPU limit for a Pod is the sum of the CPU limits for all the Containers in the Pod.*

*Pod scheduling is based on requests. A Pod is scheduled to run on a Node only if the Node has enough CPU resources available to satisfy the Pod CPU request.*

*In this exercise, you create a Pod that has a CPU request so big that it exceeds the capacity of any Node in your cluster. Here is the configuration file for a Pod that has one Container. The Container requests 100 CPU, which is likely to exceed the capacity of any Node in your cluster.*

[pods/resource/cpu-request-limit-2.yaml](https://k8s.io/examples/pods/resource/cpu-request-limit-2.yaml)



```
apiVersion: v1
kind: Pod
metadata:
  name: cpu-demo-2
  namespace: cpu-example
spec:
  containers:
    - name: cpu-demo-ctr-2
      image: vish/stress
      resources:
        limits:
          cpu: "100"
        requests:
          cpu: "100"
      args:
        - -cpus
        - "2"
```

Create the Pod:

```
kubectl apply -f https://k8s.io/examples/pods/resource/cpu-request-limit-2.yaml --namespace=cpu-example
```

View the Pod status:

```
kubectl get pod cpu-demo-2 --namespace=cpu-example
```

The output shows that the Pod status is Pending. That is, the Pod has not been scheduled to run on any Node, and it will remain in the Pending state indefinitely:

NAME	READY	STATUS	RESTARTS	AGE
cpu-demo-2	0/1	Pending	0	7m

View detailed information about the Pod, including events:

```
kubectl describe pod cpu-demo-2 --namespace=cpu-example
```

The output shows that the Container cannot be scheduled because of insufficient CPU resources on the Nodes:

<i>Events:</i>	
<i>Reason</i>	<i>Message</i>
<i>FailedScheduling</i>	<i>No nodes are available that match all of the following predicates:: Insufficient cpu (3).</i>

Delete your Pod:

```
kubectl delete pod cpu-demo-2 --namespace(cpu-example)
```

## If you do not specify a CPU limit

If you do not specify a CPU limit for a Container, then one of these situations applies:

- The Container has no upper bound on the CPU resources it can use. The Container could use all of the CPU resources available on the Node where it is running.
- The Container is running in a namespace that has a default CPU limit, and the Container is automatically assigned the default limit. Cluster administrators can use a [LimitRange](#) to specify a default value for the CPU limit.

## If you specify a CPU limit but do not specify a CPU request

If you specify a CPU limit for a Container but do not specify a CPU request, Kubernetes automatically assigns a CPU request that matches the limit. Similarly, if a Container specifies its own memory limit, but does not specify a memory request, Kubernetes automatically assigns a memory request that matches the limit.

## Motivation for CPU requests and limits

By configuring the CPU requests and limits of the Containers that run in your cluster, you can make efficient use of the CPU resources available on your cluster Nodes. By keeping a Pod CPU request low, you give the Pod a good chance of being scheduled. By having a CPU limit that is greater than the CPU request, you accomplish two things:

- *The Pod can have bursts of activity where it makes use of CPU resources that happen to be available.*
- *The amount of CPU resources a Pod can use during a burst is limited to some reasonable amount.*

## **Clean up**

*Delete your namespace:*

```
kubectl delete namespace cpu-example
```

## **What's next**

### **For app developers**

- [Assign Memory Resources to Containers and Pods](#)
- [Configure Quality of Service for Pods](#)

### **For cluster administrators**

- [Configure Default Memory Requests and Limits for a Namespace](#)
- [Configure Default CPU Requests and Limits for a Namespace](#)
- [Configure Minimum and Maximum Memory Constraints for a Namespace](#)
- [Configure Minimum and Maximum CPU Constraints for a Namespace](#)
- [Configure Memory and CPU Quotas for a Namespace](#)
- [Configure a Pod Quota for a Namespace](#)
- [Configure Quotas for API 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](#).

Last modified October 17, 2020 at 3:21 PM PST: [update kubernetes-incubator references \(a8b6551c2\)](#)

[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Create a namespace](#)
- [Specify a CPU request and a CPU limit](#)
- [CPU units](#)
- [Specify a CPU request that is too big for your Nodes](#)
- [If you do not specify a CPU limit](#)
- [If you specify a CPU limit but do not specify a CPU request](#)
- [Motivation for CPU requests and limits](#)
- [Clean up](#)
- [What's next](#)
  - [For app developers](#)
  - [For cluster administrators](#)

# Configure GMSA for Windows Pods and containers

**FEATURE STATE:** Kubernetes v1.18 [stable]

This page shows how to configure [Group Managed Service Accounts](#) (GMSA) for Pods and containers that will run on Windows nodes. Group Managed Service Accounts are a specific type of Active Directory account that provides automatic password management, simplified service principal name (SPN) management, and the ability to delegate the management to other administrators across multiple servers.

In Kubernetes, GMSA credential specs are configured at a Kubernetes cluster-wide scope as Custom Resources. Windows Pods, as well as individual containers within a Pod, can be configured to use a GMSA for domain based functions (e.g. Kerberos authentication) when interacting with other Windows services. As of v1.16, the Docker runtime supports GMSA for Windows workloads.

## Before you begin

You need to have a Kubernetes cluster and the `kubectl` command-line tool must be configured to communicate with your cluster. The cluster is expected to have Windows worker nodes. This section covers a set of initial steps required once for each cluster:

## **Install the GMSACredentialSpec CRD**

A [CustomResourceDefinition](#)(CRD) for GMSA credential spec resources needs to be configured on the cluster to define the custom resource type `GMSACredentialSpec`. Download the GMSA CRD [YAML](#) and save it as `gmsa-crd.yaml`. Next, install the CRD with `kubectl apply -f gmsa-crd.yaml`

## **Install webhooks to validate GMSA users**

Two webhooks need to be configured on the Kubernetes cluster to populate and validate GMSA credential spec references at the Pod or container level:

1. A mutating webhook that expands references to GMSAs (by name from a Pod specification) into the full credential spec in JSON form within the Pod spec.
2. A validating webhook ensures all references to GMSAs are authorized to be used by the Pod service account.

Installing the above webhooks and associated objects require the steps below:

1. Create a certificate key pair (that will be used to allow the webhook container to communicate to the cluster)
2. Install a secret with the certificate from above.
3. Create a deployment for the core webhook logic.
4. Create the validating and mutating webhook configurations referring to the deployment.

A [script](#) can be used to deploy and configure the GMSA webhooks and associated objects mentioned above. The script can be run with a `--dry-run=server` option to allow you to review the changes that would be made to your cluster.

The [YAML template](#) used by the script may also be used to deploy the webhooks and associated objects manually (with appropriate substitutions for the parameters)

## **Configure GMSAs and Windows nodes in Active Directory**

Before Pods in Kubernetes can be configured to use GMSAs, the desired GMSAs need to be provisioned in Active Directory as described in the [Windows GMSA documentation](#). Windows worker nodes (that are part of the Kubernetes cluster) need to be configured in Active Directory to access the secret credentials associated with the desired GMSA as described in the [Windows GMSA documentation](#)

## Create GMSA credential spec resources

With the `GMSACredentialSpec` CRD installed (as described earlier), custom resources containing GMSA credential specs can be configured. The GMSA credential spec does not contain secret or sensitive data. It is information that a container runtime can use to describe the desired GMSA of a container to Windows. GMSA credential specs can be generated in YAML format with a utility [PowerShell script](#).

Following are the steps for generating a GMSA credential spec YAML manually in JSON format and then converting it:

1. Import the `CredentialSpec` [module](#): `ipmo CredentialSpec.psm1`
2. Create a credential spec in JSON format using `New-CredentialSpec`. To create a GMSA credential spec named `WebApp1`, invoke `New-CredentialSpec -Name WebApp1 -AccountName WebApp1 -Domain $(Get-ADDomain -Current LocalComputer)`
3. Use `Get-CredentialSpec` to show the path of the JSON file.
4. Convert the `credspec` file from JSON to YAML format and apply the necessary header fields `apiVersion`, `kind`, `metadata` and `credspec` to make it a `GMSACredentialSpec` custom resource that can be configured in Kubernetes.

The following YAML configuration describes a GMSA credential spec named `gmsa-WebApp1`:

```
apiVersion: windows.k8s.io/v1alpha1
kind: GMSACredentialSpec
metadata:
  name: gmsa-WebApp1 #This is an arbitrary name but it will be
  used as a reference
credspec:
  ActiveDirectoryConfig:
    GroupManagedServiceAccounts:
      - Name: WebApp1 #Username of the GMSA account
        Scope: CONTOSO #NETBIOS Domain Name
      - Name: WebApp1 #Username of the GMSA account
        Scope: contoso.com #DNS Domain Name
  CmsPlugins:
    - ActiveDirectory
  DomainJoinConfig:
    DnsName: contoso.com #DNS Domain Name
    DnsTreeName: contoso.com #DNS Domain Name Root
    Guid: 244818ae-87ac-4fcf-92ec-e79e5252348a #GUID
    MachineAccountName: WebApp1 #Username of the GMSA account
    NetBiosName: CONTOSO #NETBIOS Domain Name
    Sid: S-1-5-21-2126449477-2524075714-3094792973 #SID of GMSA
```

The above credential spec resource may be saved as `gmsa-Webapp1-credspec.yaml` and applied to the cluster using: `kubectl apply -f gmsa-Webapp1-credspec.yaml`

## **Configure cluster role to enable RBAC on specific GMSA credential specs**

A cluster role needs to be defined for each GMSA credential spec resource. This authorizes the `use` verb on a specific GMSA resource by a subject which is typically a service account. The following example shows a cluster role that authorizes usage of the `gmsa-WebApp1` credential spec from above. Save the file as `gmsa-webapp1-role.yaml` and apply using `kubectl apply -f gmsa-webapp1-role.yaml`

```
#Create the Role to read the credspec
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: webapp1-role
rules:
- apiGroups: ["windows.k8s.io"]
  resources: ["gmsacredentialspecs"]
  verbs: ["use"]
  resourceNames: ["gmsa-WebApp1"]
```

## **Assign role to service accounts to use specific GMSA credspecs**

A service account (that Pods will be configured with) needs to be bound to the cluster role created above. This authorizes the service account to use the desired GMSA credential spec resource. The following shows the default service account being bound to a cluster role `webapp1-role` to use `gmsa-WebApp1` credential spec resource created above.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: allow-default-svc-account-read-on-gmsa-WebApp1
  namespace: default
subjects:
- kind: ServiceAccount
  name: default
  namespace: default
roleRef:
  kind: ClusterRole
  name: webapp1-role
  apiGroup: rbac.authorization.k8s.io
```

## **Configure GMSA credential spec reference in Pod spec**

The Pod spec field `securityContext.windowsOptions.gmsaCredentialSpecName` is used to specify references to desired GMSA credential spec custom resources in Pod specs. This configures all containers in the Pod spec to use the specified GMSA. A sample Pod spec with the annotation populated to refer to `gmsa-WebApp1`:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    run: with-creds
  name: with-creds
  namespace: default
spec:
  replicas: 1
  selector:
    matchLabels:
      run: with-creds
  template:
    metadata:
      labels:
        run: with-creds
    spec:
      securityContext:
        windowsOptions:
          gmsaCredentialSpecName: gmsa-webapp1
      containers:
        - image: mcr.microsoft.com/windows/servercore/
iis:windowsservercore-ltsc2019
          imagePullPolicy: Always
          name: iis
        nodeSelector:
          kubernetes.io/os: windows
```

Individual containers in a Pod spec can also specify the desired GMSA credspec using a per-container `securityContext.windowsOptions.gmsaCredentialSpecName` field. For example:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    run: with-creds
  name: with-creds
  namespace: default
spec:
  replicas: 1
  selector:
```

```

  matchLabels:
    run: with-creds
  template:
    metadata:
      labels:
        run: with-creds
    spec:
      containers:
        - image: mcr.microsoft.com/windows/servercore/
iis:windowsservercore-ltsc2019
          imagePullPolicy: Always
          name: iis
          securityContext:
            windowsOptions:
              gmsaCredentialSpecName: gmsa-Webapp1
          nodeSelector:
            kubernetes.io/os: windows

```

As Pod specs with GMSA fields populated (as described above) are applied in a cluster, the following sequence of events take place:

1. The mutating webhook resolves and expands all references to GMSA credential spec resources to the contents of the GMSA credential spec.
2. The validating webhook ensures the service account associated with the Pod is authorized for the use verb on the specified GMSA credential spec.
3. The container runtime configures each Windows container with the specified GMSA credential spec so that the container can assume the identity of the GMSA in Active Directory and access services in the domain using that identity.

## Troubleshooting

If you are having difficulties getting GMSA to work in your environment, there are a few troubleshooting steps you can take.

First, make sure the credspec has been passed to the Pod. To do this you will need to exec into one of your Pods and check the output of the `nltest.exe /parentdomain` command. In the example below the Pod did not get the credspec correctly:

```
kubectl exec -it iis-auth-7776966999-n5nzs powershell.exe
```

*Windows PowerShell*

*Copyright (C) Microsoft Corporation. All rights reserved.*

```
PS C:\> nltest.exe /parentdomain
Getting parent domain failed: Status = 1722 0x6ba
RPC_S_SERVER_UNAVAILABLE
PS C:\>
```

If your Pod did get the credspec correctly, then next check communication with the domain. First, from inside of your Pod, quickly do an nslookup to find the root of your domain.

This will tell us 3 things:

1. The Pod can reach the DC
2. The DC can reach the Pod
3. DNS is working correctly.

If the DNS and communication test passes, next you will need to check if the Pod has established secure channel communication with the domain. To do this, again, exec into your Pod and run the `nltest.exe /query` command.

```
PS C:\> nltest.exe /query
I_NetLogonControl failed: Status = 1722 0x6ba
RPC_S_SERVER_UNAVAILABLE
```

This tells us that for some reason, the Pod was unable to logon to the domain using the account specified in the credspec. You can try to repair the secure channel by running the `nltest.exe /sc_reset:domain.example` command.

```
PS C:\> nltest /sc_reset:domain.example
Flags: 30 HAS_IP HAS_TIMESERV
Trusted DC Name \\dc10.domain.example
Trusted DC Connection Status Status = 0 0x0 NERR_Success
The command completed successfully
PS C:\>
```

If the above command corrects the error, you can automate the step by adding the following lifecycle hook to your Pod spec. If it did not correct the error, you will need to examine your credspec again and confirm that it is correct and complete.

```
image: registry.domain.example/iis-auth:1809v1
lifecycle:
  postStart:
    exec:
      command: ["powershell.exe", "-command", "do
{ Restart-Service -Name netlogon } while ( $($Result =
 nltest.exe /query); if ($Result -like '*0x0 NERR_Success*')
{return $true} else {return $false}) -eq $false)"]
    imagePullPolicy: IfNotPresent
```

If you add the `lifecycle` section show above to your Pod spec, the Pod will execute the commands listed to restart the `netlogon` service until the `nltest.exe /query` command exits without error.

## GMSA limitations

When using the [ContainerD runtime for Windows](#) accessing restricted network shares via the GMSA domain identity fails. The container will

receive the identity of and calls from `nltest.exe /query` will work. It is recommended to use the [Docker EE runtime](#) if access to network shares is required. The Windows Server team is working on resolving the issue in the Windows Kernel and will release a patch to resolve this issue in the future. Look for updates on the [Microsoft Windows Containers issue tracker](#).

## 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](#).

Last modified August 13, 2020 at 11:06 AM PST: [Update manifests with the correct API version \(d8b3ec4f6\)](#)

[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
  - [Install the GMSACredentialSpec CRD](#)
  - [Install webhooks to validate GMSA users](#)
- [Configure GMSAs and Windows nodes in Active Directory](#)
- [Create GMSA credential spec resources](#)
- [Configure cluster role to enable RBAC on specific GMSA credential specs](#)
- [Assign role to service accounts to use specific GMSA credsspecs](#)
- [Configure GMSA credential spec reference in Pod spec](#)
- [Troubleshooting](#)
- [GMSA limitations](#)

# Configure RunAsUserName for Windows pods and containers

**FEATURE STATE:** Kubernetes v1.18 [stable]

This page shows how to use the `runAsUserName` setting for Pods and containers that will run on Windows nodes. This is roughly equivalent of the Linux-specific `runAsUser` setting, allowing you to run applications in a container as a different username than the default.

## Before you begin

You need to have a Kubernetes cluster and the `kubectl` command-line tool must be configured to communicate with your cluster. The cluster is expected to have Windows worker nodes where pods with containers running Windows workloads will get scheduled.

## **Set the Username for a Pod**

To specify the username with which to execute the Pod's container processes, include the `securityContext` field ([PodSecurityContext](#) in the Pod specification, and within it, the `windowsOptions` ([WindowsSecurityContextOptions](#)) field containing the `runAsUserName` field.

The Windows security context options that you specify for a Pod apply to all Containers and init Containers in the Pod.

Here is a configuration file for a Windows Pod that has the `runAsUserName` field set:

[`windows/run-as-username-pod.yaml`](#)  


```
apiVersion: v1
kind: Pod
metadata:
  name: run-as-username-pod-demo
spec:
  securityContext:
    windowsOptions:
      runAsUserName: "ContainerUser"
  containers:
  - name: run-as-username-demo
    image: mcr.microsoft.com/windows/servercore:ltsc2019
    command: ["ping", "-t", "localhost"]
  nodeSelector:
    kubernetes.io/os: windows
```

Create the Pod:

```
kubectl apply -f https://k8s.io/examples/windows/run-as-username-pod.yaml
```

Verify that the Pod's Container is running:

```
kubectl get pod run-as-username-pod-demo
```

Get a shell to the running Container:

```
kubectl exec -it run-as-username-pod-demo -- powershell
```

Check that the shell is running user the correct username:

```
echo $env:USERNAME
```

The output should be:

```
ContainerUser
```

# Set the Username for a Container

To specify the username with which to execute a Container's processes, include the `securityContext` field ([SecurityContext](#)) in the Container manifest, and within it, the `windowsOptions` ([WindowsSecurityContextOptions](#)) field containing the `runAsUserName` field.

The Windows security context options that you specify for a Container apply only to that individual Container, and they override the settings made at the Pod level.

Here is the configuration file for a Pod that has one Container, and the `runAsUserName` field is set at the Pod level and the Container level:

[`windows/run-as-username-container.yaml`](#)  


```
apiVersion: v1
kind: Pod
metadata:
  name: run-as-username-container-demo
spec:
  securityContext:
    windowsOptions:
      runAsUserName: "ContainerUser"
  containers:
  - name: run-as-username-demo
    image: mcr.microsoft.com/windows/servercore:ltsc2019
    command: ["ping", "-t", "localhost"]
    securityContext:
      windowsOptions:
        runAsUserName: "ContainerAdministrator"
  nodeSelector:
    kubernetes.io/os: windows
```

Create the Pod:

```
kubectl apply -f https://k8s.io/examples/windows/run-as-username-
container.yaml
```

Verify that the Pod's Container is running:

```
kubectl get pod run-as-username-container-demo
```

Get a shell to the running Container:

```
kubectl exec -it run-as-username-container-demo -- powershell
```

Check that the shell is running user the correct username (the one set at the Container level):

```
echo $env:USERNAME
```

The output should be:

`ContainerAdministrator`

## Windows Username limitations

In order to use this feature, the value set in the `runAsUserName` field must be a valid username. It must have the following format: `DOMAIN\USER`, where `DOMAIN\` is optional. Windows user names are case insensitive. Additionally, there are some restrictions regarding the `DOMAIN` and `USER`:

- The `runAsUserName` field cannot be empty, and it cannot contain control characters (ASCII values: `0x00-0x1F, 0x7F`)
- The `DOMAIN` must be either a NetBios name, or a DNS name, each with their own restrictions:
  - NetBios names: maximum 15 characters, cannot start with `.` (dot), and cannot contain the following characters: `\ / : * ? " < > |`
  - DNS names: maximum 255 characters, contains only alphanumeric characters, dots, and dashes, and it cannot start or end with a `.` (dot) or `-` (dash).
- The `USER` must have at most 20 characters, it cannot contain only dots or spaces, and it cannot contain the following characters: `" / \ [ ] : ; | = , + * ? < > @.`

Examples of acceptable values for the `runAsUserName` field: `ContainerAdministrator`, `ContainerUser`, `NT AUTHORITY\NETWORK SERVICE`, `NT AUTHORITY\LOCAL SERVICE`.

For more information about these limitations, check [here](#) and [here](#).

## What's next

- [Guide for scheduling Windows containers in Kubernetes](#)
- [Managing Workload Identity with Group Managed Service Accounts \(GMSA\)](#)
- [Configure GMSA for Windows pods and containers](#)

## 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](#).

Last modified May 30, 2020 at 3:10 PM PST: [add en pages \(ecc27bbbe\)](#)  
[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)

- [Set the Username for a Pod](#)
- [Set the Username for a Container](#)
- [Windows Username limitations](#)
- [What's next](#)

# Configure Quality of Service for Pods

This page shows how to configure Pods so that they will be assigned particular Quality of Service (QoS) classes. Kubernetes uses QoS classes to make decisions about scheduling and evicting Pods.

## Before you begin

You need to have a Kubernetes cluster, and the `kubectl` command-line tool must be configured to communicate with your cluster. If you do not already have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:

- [Katacoda](#)
- [Play with Kubernetes](#)

To check the version, enter `kubectl version`.

## QoS classes

When Kubernetes creates a Pod it assigns one of these QoS classes to the Pod:

- *Guaranteed*
- *Burstable*
- *BestEffort*

## Create a namespace

Create a namespace so that the resources you create in this exercise are isolated from the rest of your cluster.

```
kubectl create namespace qos-example
```

# Create a Pod that gets assigned a QoS class of Guaranteed

For a Pod to be given a QoS class of Guaranteed:

- Every Container, including init containers, in the Pod must have a memory limit and a memory request, and they must be the same.
- Every Container, including init containers, in the Pod must have a CPU limit and a CPU request, and they must be the same.

Here is the configuration file for a Pod that has one Container. The Container has a memory limit and a memory request, both equal to 200 MiB. The Container has a CPU limit and a CPU request, both equal to 700 milliCPU:

[pods/qos/qos-pod.yaml](#)



```
apiVersion: v1
kind: Pod
metadata:
  name: qos-demo
  namespace: qos-example
spec:
  containers:
    - name: qos-demo-ctr
      image: nginx
      resources:
        limits:
          memory: "200Mi"
          cpu: "700m"
        requests:
          memory: "200Mi"
          cpu: "700m"
```

Create the Pod:

```
kubectl apply -f https://k8s.io/examples/pods/qos/qos-pod.yaml --namespace=qos-example
```

View detailed information about the Pod:

```
kubectl get pod qos-demo --namespace=qos-example --output=yaml
```

The output shows that Kubernetes gave the Pod a QoS class of Guaranteed. The output also verifies that the Pod Container has a memory request that matches its memory limit, and it has a CPU request that matches its CPU limit.

```
spec:  
  containers:  
    ...  
      resources:  
        limits:  
          cpu: 700m  
          memory: 200Mi  
        requests:  
          cpu: 700m  
          memory: 200Mi  
    ...  
  status:  
    qosClass: Guaranteed
```

**Note:** If a Container specifies its own memory limit, but does not specify a memory request, Kubernetes automatically assigns a memory request that matches the limit. Similarly, if a Container specifies its own CPU limit, but does not specify a CPU request, Kubernetes automatically assigns a CPU request that matches the limit.

Delete your Pod:

```
kubectl delete pod qos-demo --namespace=qos-example
```

## Create a Pod that gets assigned a QoS class of Burstable

A Pod is given a QoS class of Burstable if:

- The Pod does not meet the criteria for QoS class Guaranteed.
- At least one Container in the Pod has a memory or CPU request.

Here is the configuration file for a Pod that has one Container. The Container has a memory limit of 200 MiB and a memory request of 100 MiB.

[pods/qos/qos-pod-2.yaml](#)



```
apiVersion: v1
kind: Pod
metadata:
  name: qos-demo-2
  namespace: qos-example
spec:
  containers:
    - name: qos-demo-2-ctr
      image: nginx
      resources:
        limits:
          memory: "200Mi"
        requests:
          memory: "100Mi"
```

Create the Pod:

```
kubectl apply -f https://k8s.io/examples/pods/qos/qos-pod-2.yaml
--namespace=qos-example
```

View detailed information about the Pod:

```
kubectl get pod qos-demo-2 --namespace=qos-example --output=yaml
```

The output shows that Kubernetes gave the Pod a QoS class of *Burstable*.

```
spec:
  containers:
    - image: nginx
      imagePullPolicy: Always
      name: qos-demo-2-ctr
      resources:
        limits:
          memory: 200Mi
        requests:
          memory: 100Mi
    ...
status:
  qosClass: Burstable
```

Delete your Pod:

```
kubectl delete pod qos-demo-2 --namespace=qos-example
```

# **Create a Pod that gets assigned a QoS class of BestEffort**

For a Pod to be given a QoS class of BestEffort, the Containers in the Pod must not have any memory or CPU limits or requests.

Here is the configuration file for a Pod that has one Container. The Container has no memory or CPU limits or requests:

[pods/qos/qos-pod-3.yaml](#)



```
apiVersion: v1
kind: Pod
metadata:
  name: qos-demo-3
  namespace: qos-example
spec:
  containers:
    - name: qos-demo-3-ctr
      image: nginx
```

Create the Pod:

```
kubectl apply -f https://k8s.io/examples/pods/qos/qos-pod-3.yaml
--namespace=qos-example
```

View detailed information about the Pod:

```
kubectl get pod qos-demo-3 --namespace=qos-example --output=yaml
```

The output shows that Kubernetes gave the Pod a QoS class of BestEffort.

```
spec:
  containers:
    ...
    resources: {}
  ...
status:
  qosClass: BestEffort
```

Delete your Pod:

```
kubectl delete pod qos-demo-3 --namespace=qos-example
```

## Create a Pod that has two Containers

Here is the configuration file for a Pod that has two Containers. One container specifies a memory request of 200 MiB. The other Container does not specify any requests or limits.

[pods/qos/qos-pod-4.yaml](#)



```
apiVersion: v1
kind: Pod
metadata:
  name: qos-demo-4
  namespace: qos-example
spec:
  containers:
    - name: qos-demo-4-ctr-1
      image: nginx
      resources:
        requests:
          memory: "200Mi"
    - name: qos-demo-4-ctr-2
      image: redis
```

Notice that this Pod meets the criteria for QoS class *Burstable*. That is, it does not meet the criteria for QoS class *Guaranteed*, and one of its Containers has a memory request.

Create the Pod:

```
kubectl apply -f https://k8s.io/examples/pods/qos/qos-pod-4.yaml
--namespace=qos-example
```

View detailed information about the Pod:

```
kubectl get pod qos-demo-4 --namespace=qos-example --output=yaml
```

The output shows that Kubernetes gave the Pod a QoS class of *Burstable*:

```
spec:  
  containers:  
    ..  
      name: qos-demo-4-ctr-1  
      resources:  
        requests:  
          memory: 200Mi  
    ..  
      name: qos-demo-4-ctr-2  
      resources: {}  
    ..  
  status:  
    qosClass: Burstable
```

*Delete your Pod:*

```
kubectl delete pod qos-demo-4 --namespace=qos-example
```

## Clean up

*Delete your namespace:*

```
kubectl delete namespace qos-example
```

## What's next

### For app developers

- [Assign Memory Resources to Containers and Pods](#)
- [Assign CPU Resources to Containers and Pods](#)

### For cluster administrators

- [Configure Default Memory Requests and Limits for a Namespace](#)
- [Configure Default CPU Requests and Limits for a Namespace](#)
- [Configure Minimum and Maximum Memory Constraints for a Namespace](#)
- [Configure Minimum and Maximum CPU Constraints for a Namespace](#)

## [Configure Memory and CPU Quotas for a Namespace](#)

- [Configure a Pod Quota for a Namespace](#)
- [Configure Quotas for API Objects](#)
- [Control Topology Management policies on a node](#)

## **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](#).

Last modified November 06, 2020 at 7:19 PM PST: [Explicitly mention init containers in the QoS doc \(75456274f\)](#)

[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [QoS classes](#)
- [Create a namespace](#)
- [Create a Pod that gets assigned a QoS class of Guaranteed](#)
- [Create a Pod that gets assigned a QoS class of Burstable](#)
- [Create a Pod that gets assigned a QoS class of BestEffort](#)
- [Create a Pod that has two Containers](#)
- [Clean up](#)
- [What's next](#)
  - [For app developers](#)
  - [For cluster administrators](#)

# **Assign Extended Resources to a Container**

**FEATURE STATE:** Kubernetes v1.20 [stable]

This page shows how to assign extended resources to a Container.

## **Before you begin**

You need to have a Kubernetes cluster, and the `kubectl` command-line tool must be configured to communicate with your cluster. If you do not already

have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:

- [Katacoda](#)
- [Play with Kubernetes](#)

To check the version, enter `kubectl version`.

Before you do this exercise, do the exercise in [Advertise Extended Resources for a Node](#). That will configure one of your Nodes to advertise a dongle resource.

## Assign an extended resource to a Pod

To request an extended resource, include the `resources.requests` field in your Container manifest. Extended resources are fully qualified with any domain outside of `*.kubernetes.io/`. Valid extended resource names have the form `example.com/foo` where `example.com` is replaced with your organization's domain and `foo` is a descriptive resource name.

Here is the configuration file for a Pod that has one Container:

[pods/resource/extended-resource-pod.yaml](#)  


```
apiVersion: v1
kind: Pod
metadata:
  name: extended-resource-demo
spec:
  containers:
    - name: extended-resource-demo-ctr
      image: nginx
      resources:
        requests:
          example.com/dongle: 3
        limits:
          example.com/dongle: 3
```

In the configuration file, you can see that the Container requests 3 dongles.

Create a Pod:

```
kubectl apply -f https://k8s.io/examples/pods/resource/extended-
resource-pod.yaml
```

*Verify that the Pod is running:*

```
kubectl get pod extended-resource-demo
```

*Describe the Pod:*

```
kubectl describe pod extended-resource-demo
```

*The output shows dongle requests:*

```
Limits:  
  example.com/dongle: 3  
Requests:  
  example.com/dongle: 3
```

## **Attempt to create a second Pod**

*Here is the configuration file for a Pod that has one Container. The Container requests two dongles.*

[pods/resource/extended-resource-pod-2.yaml](#)  


```
apiVersion: v1  
kind: Pod  
metadata:  
  name: extended-resource-demo-2  
spec:  
  containers:  
    - name: extended-resource-demo-2-ctr  
      image: nginx  
      resources:  
        requests:  
          example.com/dongle: 2  
        limits:  
          example.com/dongle: 2
```

*Kubernetes will not be able to satisfy the request for two dongles, because the first Pod used three of the four available dongles.*

*Attempt to create a Pod:*

```
kubectl apply -f https://k8s.io/examples/pods/resource/extended-resource-pod-2.yaml
```

Describe the Pod

```
kubectl describe pod extended-resource-demo-2
```

The output shows that the Pod cannot be scheduled, because there is no Node that has 2 dongles available:

Conditions:

Type	Status
PodScheduled	False

...

Events:

...	Warning FailedScheduling	pod (extended-resource-demo-2) failed to fit in any node
	fit failure summary on nodes :	Insufficient example.com/dongle (1)

View the Pod status:

```
kubectl get pod extended-resource-demo-2
```

The output shows that the Pod was created, but not scheduled to run on a Node. It has a status of Pending:

NAME	READY	STATUS	RESTARTS	AGE
extended-resource-demo-2	0/1	Pending	0	6m

## Clean up

Delete the Pods that you created for this exercise:

```
kubectl delete pod extended-resource-demo  
kubectl delete pod extended-resource-demo-2
```

## What's next

## **For application developers**

- [Assign Memory Resources to Containers and Pods](#)
- [Assign CPU Resources to Containers and Pods](#)

## **For cluster administrators**

- [Advertise Extended Resources for a Node](#)

## **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](#).

Last modified May 30, 2020 at 3:10 PM PST: [add en pages \(ecc27bbbe\)](#)  
[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Assign an extended resource to a Pod](#)
- [Attempt to create a second Pod](#)
- [Clean up](#)
- [What's next](#)
  - [For application developers](#)
  - [For cluster administrators](#)

# **Configure a Pod to Use a Volume for Storage**

This page shows how to configure a Pod to use a Volume for storage.

A Container's file system lives only as long as the Container does. So when a Container terminates and restarts, filesystem changes are lost. For more consistent storage that is independent of the Container, you can use a [Volume](#). This is especially important for stateful applications, such as key-value stores (such as Redis) and databases.

## **Before you begin**

You need to have a Kubernetes cluster, and the `kubectl` command-line tool must be configured to communicate with your cluster. If you do not already have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:

- [Katacoda](#)
- [Play with Kubernetes](#)

To check the version, enter `kubectl version`.

## Configure a volume for a Pod

In this exercise, you create a Pod that runs one Container. This Pod has a Volume of type [emptyDir](#) that lasts for the life of the Pod, even if the Container terminates and restarts. Here is the configuration file for the Pod:

[`pods/storage/redis.yaml`](#)



```
apiVersion: v1
kind: Pod
metadata:
  name: redis
spec:
  containers:
    - name: redis
      image: redis
      volumeMounts:
        - name: redis-storage
          mountPath: /data/redis
  volumes:
    - name: redis-storage
      emptyDir: {}
```

1. Create the Pod:

```
kubectl apply -f https://k8s.io/examples/pods/storage/redis.yaml
```

2. Verify that the Pod's Container is running, and then watch for changes to the Pod:

```
kubectl get pod redis --watch
```

The output looks like this:

NAME	READY	STATUS	RESTARTS	AGE
redis	1/1	Running	0	13s

3. In another terminal, get a shell to the running Container:

```
kubectl exec -it redis -- /bin/bash
```

4. In your shell, go to `/data/redis`, and then create a file:

```
root@redis:/data# cd /data/redis/  
root@redis:/data/redis# echo Hello > test-file
```

5. In your shell, list the running processes:

```
root@redis:/data/redis# apt-get update  
root@redis:/data/redis# apt-get install procps  
root@redis:/data/redis# ps aux
```

The output is similar to this:

USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START
TIME	COMMAND							
redis	1	0.1	0.1	33308	3828	?	Ssl	00:46
0:00	redis-server	*	:6379					
root	12	0.0	0.0	20228	3020	?	Ss	00:47
0:00	/bin/bash							
root	15	0.0	0.0	17500	2072	?	R+	00:48
0:00	ps aux							

6. In your shell, kill the Redis process:

```
root@redis:/data/redis# kill <pid>
```

where `<pid>` is the Redis process ID (PID).

7. In your original terminal, watch for changes to the Redis Pod. Eventually, you will see something like this:

NAME	READY	STATUS	RESTARTS	AGE
redis	1/1	Running	0	13s
redis	0/1	Completed	0	6m
redis	1/1	Running	1	6m

At this point, the Container has terminated and restarted. This is because the Redis Pod has a [restartPolicy](#) of Always.

1. Get a shell into the restarted Container:

```
kubectl exec -it redis -- /bin/bash
```

2. In your shell, go to `/data/redis`, and verify that `test-file` is still there.

```
root@redis:/data/redis# cd /data/redis/  
root@redis:/data/redis# ls  
test-file
```

3. Delete the Pod that you created for this exercise:

```
kubectl delete pod redis
```

## What's next

- See [Volume](#).
- See [Pod](#).
- In addition to the local disk storage provided by `emptyDir`, Kubernetes supports many different network-attached storage solutions, including PD on GCE and EBS on EC2, which are preferred for critical data and will handle details such as mounting and unmounting the devices on the nodes. See [Volumes](#) for more details.

## 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](#).

Last modified May 30, 2020 at 3:10 PM PST: [add en pages \(ecc27bbbe\)](#)  
[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Configure a volume for a Pod](#)
- [What's next](#)

# Configure a Pod to Use a PersistentVolume for Storage

This page shows you how to configure a Pod to use a [PersistentVolumeClaim](#) for storage. Here is a summary of the process:

1. You, as cluster administrator, create a `PersistentVolume` backed by physical storage. You do not associate the volume with any Pod.
2. You, now taking the role of a developer / cluster user, create a `PersistentVolumeClaim` that is automatically bound to a suitable `PersistentVolume`.

- You create a Pod that uses the above PersistentVolumeClaim for 3. storage.

## Before you begin

- You need to have a Kubernetes cluster that has only one Node, and the [kubectl](#) command-line tool must be configured to communicate with your cluster. If you do not already have a single-node cluster, you can create one by using [Minikube](#).
- Familiarize yourself with the material in [Persistent Volumes](#).

## Create an index.html file on your Node

Open a shell to the single Node in your cluster. How you open a shell depends on how you set up your cluster. For example, if you are using Minikube, you can open a shell to your Node by entering `minikube ssh`.

In your shell on that Node, create a `/mnt/data` directory:

```
# This assumes that your Node uses "sudo" to run commands
# as the superuser
sudo mkdir /mnt/data
```

In the `/mnt/data` directory, create an `index.html` file:

```
# This again assumes that your Node uses "sudo" to run commands
# as the superuser
sudo sh -c "echo 'Hello from Kubernetes storage' > /mnt/data/
index.html"
```

**Note:** If your Node uses a tool for superuser access other than `sudo`, you can usually make this work if you replace `sudo` with the name of the other tool.

Test that the `index.html` file exists:

```
cat /mnt/data/index.html
```

The output should be:

```
Hello from Kubernetes storage
```

You can now close the shell to your Node.

## Create a PersistentVolume

In this exercise, you create a hostPath PersistentVolume. Kubernetes supports hostPath for development and testing on a single-node cluster. A hostPath PersistentVolume uses a file or directory on the Node to emulate network-attached storage.

*In a production cluster, you would not use hostPath. Instead a cluster administrator would provision a network resource like a Google Compute Engine persistent disk, an NFS share, or an Amazon Elastic Block Store volume. Cluster administrators can also use [StorageClasses](#) to set up dynamic provisioning.*

*Here is the configuration file for the hostPath PersistentVolume:*

[pods/storage/pv-volume.yaml](#)



```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: task-pv-volume
  labels:
    type: local
spec:
  storageClassName: manual
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: "/mnt/data"
```

*The configuration file specifies that the volume is at /mnt/data on the cluster's Node. The configuration also specifies a size of 10 gibibytes and an access mode of ReadWriteOnce, which means the volume can be mounted as read-write by a single Node. It defines the [StorageClass name](#) manual for the PersistentVolume, which will be used to bind PersistentVolumeClaim requests to this PersistentVolume.*

*Create the PersistentVolume:*

```
kubectl apply -f https://k8s.io/examples/pods/storage/pv-volume.yaml
```

*View information about the PersistentVolume:*

```
kubectl get pv task-pv-volume
```

*The output shows that the PersistentVolume has a STATUS of Available. This means it has not yet been bound to a PersistentVolumeClaim.*

NAME	CAPACITY	ACCESSMODES	RECLAIMPOLICY	
STATUS	CLAIM	STORAGECLASS	REASON	AGE
task-pv-volume	10Gi	RWO	Retain	
Available		manual		4s

# Create a PersistentVolumeClaim

The next step is to create a `PersistentVolumeClaim`. Pods use `PersistentVolumeClaims` to request physical storage. In this exercise, you create a `PersistentVolumeClaim` that requests a volume of at least three gibibytes that can provide read-write access for at least one Node.

Here is the configuration file for the `PersistentVolumeClaim`:

[`pods/storage/pv-claim.yaml`](#)



```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: task-pv-claim
spec:
  storageClassName: manual
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 3Gi
```

Create the `PersistentVolumeClaim`:

```
kubectl apply -f https://k8s.io/examples/pods/storage/pv-claim.yaml
```

After you create the `PersistentVolumeClaim`, the Kubernetes control plane looks for a `PersistentVolume` that satisfies the claim's requirements. If the control plane finds a suitable `PersistentVolume` with the same `StorageClass`, it binds the claim to the volume.

Look again at the `PersistentVolume`:

```
kubectl get pv task-pv-volume
```

Now the output shows a `STATUS` of `Bound`.

NAME	CAPACITY	ACCESSMODES	RECLAIMPOLICY	AGE
STATUS	CLAIM	STORAGECLASS	REASON	
task-pv-volume	10Gi	RWO	Retain	
Bound	default/task-pv-claim	manual		2m

Look at the `PersistentVolumeClaim`:

```
kubectl get pvc task-pv-claim
```

The output shows that the `PersistentVolumeClaim` is bound to your `PersistentVolume`, `task-pv-volume`.

NAME	STATUS	VOLUME	CAPACITY
ACCESSMODES	STORAGECLASS	AGE	
task-pv-claim	Bound	task-pv-volume	10Gi
RwO	manual	30s	

## Create a Pod

The next step is to create a *Pod* that uses your *PersistentVolumeClaim* as a volume.

Here is the configuration file for the *Pod*:

[pods/storage/pv-pod.yaml](#)



```
apiVersion: v1
kind: Pod
metadata:
  name: task-pv-pod
spec:
  volumes:
    - name: task-pv-storage
      persistentVolumeClaim:
        claimName: task-pv-claim
  containers:
    - name: task-pv-container
      image: nginx
      ports:
        - containerPort: 80
          name: "http-server"
      volumeMounts:
        - mountPath: "/usr/share/nginx/html"
          name: task-pv-storage
```

Notice that the *Pod*'s configuration file specifies a *PersistentVolumeClaim*, but it does not specify a *PersistentVolume*. From the *Pod*'s point of view, the claim is a volume.

Create the *Pod*:

```
kubectl apply -f https://k8s.io/examples/pods/storage/pv-pod.yaml
```

Verify that the container in the *Pod* is running;

```
kubectl get pod task-pv-pod
```

Get a shell to the container running in your *Pod*:

```
kubectl exec -it task-pv-pod -- /bin/bash
```

In your shell, verify that `nginx` is serving the `index.html` file from the `hostPath` volume:

```
# Be sure to run these 3 commands inside the root shell that
comes from
# running "kubectl exec" in the previous step
apt update
apt install curl
curl http://localhost/
```

The output shows the text that you wrote to the `index.html` file on the `hostPath` volume:

```
Hello from Kubernetes storage
```

If you see that message, you have successfully configured a Pod to use storage from a `PersistentVolumeClaim`.

## Clean up

Delete the Pod, the `PersistentVolumeClaim` and the `PersistentVolume`:

```
kubectl delete pod task-pv-pod
kubectl delete pvc task-pv-claim
kubectl delete pv task-pv-volume
```

If you don't already have a shell open to the Node in your cluster, open a new shell the same way that you did earlier.

In the shell on your Node, remove the file and directory that you created:

```
# This assumes that your Node uses "sudo" to run commands
# as the superuser
sudo rm /mnt/data/index.html
sudo rmdir /mnt/data
```

You can now close the shell to your Node.

## Access control

Storage configured with a group ID (GID) allows writing only by Pods using the same GID. Mismatched or missing GIDs cause permission denied errors. To reduce the need for coordination with users, an administrator can annotate a `PersistentVolume` with a GID. Then the GID is automatically added to any Pod that uses the `PersistentVolume`.

Use the `pv.beta.kubernetes.io/gid` annotation as follows:

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pvl
```

`annotations:`

`pv.beta.kubernetes.io/gid: "1234"`

When a Pod consumes a PersistentVolume that has a GID annotation, the annotated GID is applied to all containers in the Pod in the same way that GIDs specified in the Pod's security context are. Every GID, whether it originates from a PersistentVolume annotation or the Pod's specification, is applied to the first process run in each container.

**Note:** When a Pod consumes a PersistentVolume, the GIDs associated with the PersistentVolume are not present on the Pod resource itself.

## What's next

- Learn more about [PersistentVolumes](#).
- Read the [Persistent Storage design document](#).

## Reference

- [PersistentVolume](#)
- [PersistentVolumeSpec](#)
- [PersistentVolumeClaim](#)
- [PersistentVolumeClaimSpec](#)

## 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](#).

Last modified October 22, 2020 at 3:01 PM PST: [Fix links in the tasks section \(740eb340d\)](#)

[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Create an index.html file on your Node](#)
- [Create a PersistentVolume](#)
- [Create a PersistentVolumeClaim](#)
- [Create a Pod](#)
- [Clean up](#)
- [Access control](#)
- [What's next](#)
  - [Reference](#)

# Configure a Pod to Use a Projected Volume for Storage

This page shows how to use a [projected](#) Volume to mount several existing volume sources into the same directory. Currently, `secret`, `configMap`, `downwardAPI`, and `serviceAccountToken` volumes can be projected.

**Note:** `serviceAccountToken` is not a volume type.

## Before you begin

You need to have a Kubernetes cluster, and the `kubectl` command-line tool must be configured to communicate with your cluster. If you do not already have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:

- [Katacoda](#)
- [Play with Kubernetes](#)

To check the version, enter `kubectl version`.

## Configure a projected volume for a pod

In this exercise, you create username and password [Secrets](#) from local files. You then create a Pod that runs one container, using a [projected](#) Volume to mount the Secrets into the same shared directory.

Here is the configuration file for the Pod:

[pods/storage/projected.yaml](#)



```
apiVersion: v1
kind: Pod
metadata:
  name: test-projected-volume
spec:
  containers:
    - name: test-projected-volume
      image: busybox
      args:
        - sleep
        - "86400"
      volumeMounts:
        - name: all-in-one
          mountPath: "/projected-volume"
          readOnly: true
```

```
volumes:
- name: all-in-one
projected:
  sources:
    - secret:
      name: user
    - secret:
      name: pass
```

### 1. Create the Secrets:

```
# Create files containing the username and password:
echo -n "admin" > ./username.txt
echo -n "1f2d1e2e67df" > ./password.txt

# Package these files into secrets:
kubectl create secret generic user --from-file=./username.txt
kubectl create secret generic pass --from-file=./password.txt
```

### 2. Create the Pod:

```
kubectl apply -f https://k8s.io/examples/pods/storage/
projected.yaml
```

### 3. Verify that the Pod's container is running, and then watch for changes to the Pod:

```
kubectl get --watch pod test-projected-volume
```

The output looks like this:

NAME	READY	STATUS	RESTARTS	AGE
test-projected-volume	1/1	Running	0	14s

### 4. In another terminal, get a shell to the running container:

```
kubectl exec -it test-projected-volume -- /bin/sh
```

### 5. In your shell, verify that the `projected-volume` directory contains your projected sources:

```
ls /projected-volume/
```

## Clean up

Delete the Pod and the Secrets:

```
kubectl delete pod test-projected-volume
kubectl delete secret user pass
```

## What's next

- Learn more about [projected](#) volumes.
- Read the [all-in-one volume](#) design document.

## 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](#).

Last modified May 30, 2020 at 3:10 PM PST: [add en pages \(ecc27bbbe\)](#)  
[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Configure a projected volume for a pod](#)
- [Clean up](#)
- [What's next](#)

# Configure a Security Context for a Pod or Container

A security context defines privilege and access control settings for a Pod or Container. Security context settings include, but are not limited to:

- Discretionary Access Control: Permission to access an object, like a file, is based on [user ID \(UID\) and group ID \(GID\)](#).
- [Security Enhanced Linux \(SELinux\)](#): Objects are assigned security labels.
- Running as privileged or unprivileged.
- [Linux Capabilities](#): Give a process some privileges, but not all the privileges of the root user.
- [AppArmor](#): Use program profiles to restrict the capabilities of individual programs.
- [Seccomp](#): Filter a process's system calls.
- AllowPrivilegeEscalation: Controls whether a process can gain more privileges than its parent process. This bool directly controls whether

the [`no\_new\_privs`](#) flag gets set on the container process. `AllowPrivilegeEscalation` is true always when the container is: 1) run as Privileged OR 2) has `CAP_SYS_ADMIN`.

- `readOnlyRootFilesystem`: Mounts the container's root filesystem as read-only.

The above bullets are not a complete set of security context settings -- please see [`SecurityContext`](#) for a comprehensive list.

For more information about security mechanisms in Linux, see [`Overview of Linux Kernel Security Features`](#)

## Before you begin

You need to have a Kubernetes cluster, and the `kubectl` command-line tool must be configured to communicate with your cluster. If you do not already have a cluster, you can create one by using [`minikube`](#) or you can use one of these Kubernetes playgrounds:

- [`Katacoda`](#)
- [`Play with Kubernetes`](#)

To check the version, enter `kubectl version`.

## Set the security context for a Pod

To specify security settings for a Pod, include the `securityContext` field in the Pod specification. The `securityContext` field is a [`PodSecurityContext`](#) object. The security settings that you specify for a Pod apply to all Containers in the Pod. Here is a configuration file for a Pod that has a `securityContext` and an `emptyDir` volume:

[`pods/security/security-context.yaml`](#)  


```
apiVersion: v1
kind: Pod
metadata:
  name: security-context-demo
spec:
  securityContext:
    runAsUser: 1000
    runAsGroup: 3000
    fsGroup: 2000
  volumes:
  - name: sec-ctx-vol
    emptyDir: {}
  containers:
  - name: sec-ctx-demo
```

```
image: busybox
command: [ "sh", "-c", "sleep 1h" ]
volumeMounts:
- name: sec-ctx-vol
  mountPath: /data/demo
securityContext:
  allowPrivilegeEscalation: false
```

In the configuration file, the `runAsUser` field specifies that for any Containers in the Pod, all processes run with user ID 1000. The `runAsGroup` field specifies the primary group ID of 3000 for all processes within any containers of the Pod. If this field is omitted, the primary group ID of the containers will be root(0). Any files created will also be owned by user 1000 and group 3000 when `runAsGroup` is specified. Since `fsGroup` field is specified, all processes of the container are also part of the supplementary group ID 2000. The owner for volume `/data/demo` and any files created in that volume will be Group ID 2000.

Create the Pod:

```
kubectl apply -f https://k8s.io/examples/pods/security/security-context.yaml
```

Verify that the Pod's Container is running:

```
kubectl get pod security-context-demo
```

Get a shell to the running Container:

```
kubectl exec -it security-context-demo -- sh
```

In your shell, list the running processes:

```
ps
```

The output shows that the processes are running as user 1000, which is the value of `runAsUser`:

PID	USER	TIME	COMMAND
1	1000	0:00	sleep 1h
6	1000	0:00	sh
...			

*In your shell, navigate to /data, and list the one directory:*

```
cd /data  
ls -l
```

*The output shows that the /data/demo directory has group ID 2000, which is the value of fsGroup.*

```
drwxrwsrwx 2 root 2000 4096 Jun 6 20:08 demo
```

*In your shell, navigate to /data/demo, and create a file:*

```
cd demo  
echo hello > testfile
```

*List the file in the /data/demo directory:*

```
ls -l
```

*The output shows that testfile has group ID 2000, which is the value of fsGroup.*

```
-rw-r--r-- 1 1000 2000 6 Jun 6 20:08 testfile
```

*Run the following command:*

```
$ id  
uid=1000 gid=3000 groups=2000
```

*You will see that gid is 3000 which is same as runAsGroup field. If the runAsGroup was omitted the gid would remain as 0(root) and the process will be able to interact with files that are owned by root(0) group and that have the required group permissions for root(0) group.*

*Exit your shell:*

```
exit
```

# **Configure volume permission and ownership change policy for Pods**

**FEATURE STATE:** Kubernetes v1.20 [beta]

By default, Kubernetes recursively changes ownership and permissions for the contents of each volume to match the `fsGroup` specified in a Pod's `securityContext` when that volume is mounted. For large volumes, checking and changing ownership and permissions can take a lot of time, slowing Pod startup. You can use the `fsGroupChangePolicy` field inside a `securityContext` to control the way that Kubernetes checks and manages ownership and permissions for a volume.

**fsGroupChangePolicy** - `fsGroupChangePolicy` defines behavior for changing ownership and permission of the volume before being exposed inside a Pod. This field only applies to volume types that support `fsGroup` controlled ownership and permissions. This field has two possible values:

- `OnRootMismatch`: Only change permissions and ownership if permission and ownership of root directory does not match with expected permissions of the volume. This could help shorten the time it takes to change ownership and permission of a volume.
- `Always`: Always change permission and ownership of the volume when volume is mounted.

For example:

```
securityContext:  
  runAsUser: 1000  
  runAsGroup: 3000  
  fsGroup: 2000  
  fsGroupChangePolicy: "OnRootMismatch"
```

This is an alpha feature. To use it, enable the [feature gate ConfigurableFSGroupPolicy](#) for the kube-api-server, the kube-controller-manager, and for the kubelet.

**Note:** This field has no effect on ephemeral volume types such as [secret](#), [configMap](#), and [emptydir](#).

## **Set the security context for a Container**

To specify security settings for a Container, include the `securityContext` field in the Container manifest. The `securityContext` field is a [SecurityContext](#) object. Security settings that you specify for a Container apply only to the individual Container, and they override settings made at the Pod level when there is overlap. Container settings do not affect the Pod's Volumes.

Here is the configuration file for a Pod that has one Container. Both the Pod and the Container have a `securityContext` field:

[`pods/security/security-context-2.yaml`](#)  


```
apiVersion: v1
kind: Pod
metadata:
  name: security-context-demo-2
spec:
  securityContext:
    runAsUser: 1000
  containers:
    - name: sec-ctx-demo-2
      image: gcr.io/google-samples/node-hello:1.0
      securityContext:
        runAsUser: 2000
      allowPrivilegeEscalation: false
```

Create the Pod:

```
kubectl apply -f https://k8s.io/examples/pods/security/security-
context-2.yaml
```

Verify that the Pod's Container is running:

```
kubectl get pod security-context-demo-2
```

Get a shell into the running Container:

```
kubectl exec -it security-context-demo-2 -- sh
```

In your shell, list the running processes:

```
ps aux
```

The output shows that the processes are running as user 2000. This is the value of `runAsUser` specified for the Container. It overrides the value 1000 that is specified for the Pod.

USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START	TIME
<i>COMMAND</i>									
2000	1	0.0	0.0	4336	764	?	Ss	20:36	
0:00	/bin/sh	-c	node server.js						
2000	8	0.1	0.5	772124	22604	?	S1	20:36	0:00
node server.js									
...									

Exit your shell:

```
exit
```

## Set capabilities for a Container

With [Linux capabilities](#), you can grant certain privileges to a process without granting all the privileges of the root user. To add or remove Linux capabilities for a Container, include the `capabilities` field in the `securityContext` section of the Container manifest.

First, see what happens when you don't include a `capabilities` field. Here is configuration file that does not add or remove any Container capabilities:

[`pods/security/security-context-3.yaml`](#)  


```
apiVersion: v1
kind: Pod
metadata:
  name: security-context-demo-3
spec:
  containers:
    - name: sec-ctx-3
      image: gcr.io/google-samples/node-hello:1.0
```

Create the Pod:

```
kubectl apply -f https://k8s.io/examples/pods/security/security-context-3.yaml
```

Verify that the Pod's Container is running:

```
kubectl get pod security-context-demo-3
```

Get a shell into the running Container:

```
kubectl exec -it security-context-demo-3 -- sh
```

In your shell, list the running processes:

```
ps aux
```

The output shows the process IDs (PIDs) for the Container:

USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START	TIME	COMMAND
root	1	0.0	0.0	4336	796	?	Ss	18:17	0:00	/bin/sh
-c node	server.js									
root	5	0.1	0.5	772124	22700	?	Sl	18:17	0:00	node
	server.js									

In your shell, view the status for process 1:

```
cd /proc/1  
cat status
```

The output shows the capabilities bitmap for the process:

```
...  
CapPrm: 00000000a80425fb  
CapEff: 00000000a80425fb  
...
```

Make a note of the capabilities bitmap, and then exit your shell:

```
exit
```

Next, run a Container that is the same as the preceding container, except that it has additional capabilities set.

Here is the configuration file for a Pod that runs one Container. The configuration adds the CAP\_NET\_ADMIN and CAP\_SYS\_TIME capabilities:

[pods/security/security-context-4.yaml](#)



```
apiVersion: v1
kind: Pod
metadata:
  name: security-context-demo-4
spec:
  containers:
    - name: sec-ctx-4
      image: gcr.io/google-samples/node-hello:1.0
      securityContext:
        capabilities:
          add: ["NET_ADMIN", "SYS_TIME"]
```

*Create the Pod:*

```
kubectl apply -f https://k8s.io/examples/pods/security/security-context-4.yaml
```

*Get a shell into the running Container:*

```
kubectl exec -it security-context-demo-4 -- sh
```

*In your shell, view the capabilities for process 1:*

```
cd /proc/1
cat status
```

*The output shows capabilities bitmap for the process:*

```
...
CapPrm: 00000000aa0435fb
CapEff: 00000000aa0435fb
...
```

*Compare the capabilities of the two Containers:*

```
00000000a80425fb
00000000aa0435fb
```

*In the capability bitmap of the first container, bits 12 and 25 are clear. In the second container, bits 12 and 25 are set. Bit 12 is CAP\_NET\_ADMIN, and bit 25 is CAP\_SYS\_TIME. See [capability.h](#) for definitions of the capability constants.*

**Note:** Linux capability constants have the form CAP\_XXX. But when you list capabilities in your Container manifest, you must omit the CAP\_ portion of the constant. For example, to add CAP\_SYS\_TIME, include SYS\_TIME in your list of capabilities.

## **Set the Seccomp Profile for a Container**

*To set the Seccomp profile for a Container, include the seccompProfile field in the securityContext section of your Pod or Container manifest. The seccompProfile field is a [SeccompProfile](#) object consisting of type and localhostProfile. Valid options for type include RuntimeDefault, Unconfined, and Localhost. localhostProfile must only be set if type: Localhost. It indicates the path of the pre-configured profile on the node, relative to the kubelet's configured Seccomp profile location (configured with the --root-dir flag).*

*Here is an example that sets the Seccomp profile to the node's container runtime default profile:*

```
...
  securityContext:
    seccompProfile:
      type: RuntimeDefault
```

*Here is an example that sets the Seccomp profile to a pre-configured file at <kubelet-root-dir>/seccomp/my-profiles/profile-allow.json:*

```
...
  securityContext:
    seccompProfile:
      type: Localhost
      localhostProfile: my-profiles/profile-allow.json
```

## **Assign SELinux labels to a Container**

*To assign SELinux labels to a Container, include the seLinuxOptions field in the securityContext section of your Pod or Container manifest. The seLinuxOptions field is an [SELinuxOptions](#) object. Here's an example that applies an SELinux level:*

```
...  
  securityContext:  
    seLinuxOptions:  
      level: "s0:c123,c456"
```

**Note:** To assign SELinux labels, the SELinux security module must be loaded on the host operating system.

## Discussion

The security context for a Pod applies to the Pod's Containers and also to the Pod's Volumes when applicable. Specifically `fsGroup` and `seLinuxOptions` are applied to Volumes as follows:

- `fsGroup`: Volumes that support ownership management are modified to be owned and writable by the GID specified in `fsGroup`. See the [Ownership Management design document](#) for more details.
- `seLinuxOptions`: Volumes that support SELinux labeling are relabeled to be accessible by the label specified under `seLinuxOptions`. Usually you only need to set the `level` section. This sets the [Multi-Category Security \(MCS\)](#) label given to all Containers in the Pod as well as the Volumes.

**Warning:** After you specify an MCS label for a Pod, all Pods with the same label can access the Volume. If you need inter-Pod protection, you must assign a unique MCS label to each Pod.

## Clean up

Delete the Pod:

```
kubectl delete pod security-context-demo  
kubectl delete pod security-context-demo-2  
kubectl delete pod security-context-demo-3  
kubectl delete pod security-context-demo-4
```

## What's next

- [PodSecurityContext](#)
- [SecurityContext](#)
- [Tuning Docker with the newest security enhancements](#)

- [Security Contexts design document](#)
- [Ownership Management design document](#)
- [Pod Security Policies](#)
- [AllowPrivilegeEscalation design document](#)

## 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](#).

Last modified November 05, 2020 at 7:50 PM PST: [placeholder CL for fsgroup policy beta \(c5ffbec1e\)](#)

[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Set the security context for a Pod](#)
- [Configure volume permission and ownership change policy for Pods](#)
- [Set the security context for a Container](#)
- [Set capabilities for a Container](#)
- [Set the Seccomp Profile for a Container](#)
- [Assign SELinux labels to a Container](#)
- [Discussion](#)
- [Clean up](#)
- [What's next](#)

# Configure Service Accounts for Pods

A service account provides an identity for processes that run in a Pod.

**Note:** This document is a user introduction to Service Accounts and describes how service accounts behave in a cluster set up as recommended by the Kubernetes project. Your cluster administrator may have customized the behavior in your cluster, in which case this documentation may not apply.

When you (a human) access the cluster (for example, using `kubectl`), you are authenticated by the apiserver as a particular User Account (currently this is usually `admin`, unless your cluster administrator has customized your cluster). Processes in containers inside pods can also contact the apiserver. When they do, they are authenticated as a particular Service Account (for example, `default`).

## **Before you begin**

You need to have a Kubernetes cluster, and the `kubectl` command-line tool must be configured to communicate with your cluster. If you do not already have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:

- [Katacoda](#)
- [Play with Kubernetes](#)

To check the version, enter `kubectl version`.

## **Use the Default Service Account to access the API server.**

When you create a pod, if you do not specify a service account, it is automatically assigned the default service account in the same namespace. If you get the raw json or yaml for a pod you have created (for example, `kubectl get pods/<podname> -o yaml`), you can see the `spec.serviceAccountName` field has been [automatically set](#).

You can access the API from inside a pod using automatically mounted service account credentials, as described in [Accessing the Cluster](#). The API permissions of the service account depend on the [authorization plugin and policy](#) in use.

In version 1.6+, you can opt out of automounting API credentials for a service account by setting `automountServiceAccountToken: false` on the service account:

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: build-robot
automountServiceAccountToken: false
...
```

In version 1.6+, you can also opt out of automounting API credentials for a particular pod:

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
spec:
  serviceAccountName: build-robot
```

```
automountServiceAccountToken: false
```

The pod spec takes precedence over the service account if both specify a `automountServiceAccountToken` value.

## Use Multiple Service Accounts.

Every namespace has a default service account resource called `default`. You can list this and any other serviceAccount resources in the namespace with this command:

```
kubectl get serviceaccounts
```

The output is similar to this:

NAME	SECRETS	AGE
default	1	1d

You can create additional ServiceAccount objects like this:

```
kubectl apply -f - <<EOF
apiVersion: v1
kind: ServiceAccount
metadata:
  name: build-robot
EOF
```

The name of a ServiceAccount object must be a valid [DNS subdomain name](#).

If you get a complete dump of the service account object, like this:

```
kubectl get serviceaccounts/build-robot -o yaml
```

The output is similar to this:

```
apiVersion: v1
kind: ServiceAccount
metadata:
  creationTimestamp: 2015-06-16T00:12:59Z
  name: build-robot
```

```
namespace: default
resourceVersion: "272500"
uid: 721ab723-13bc-11e5-aec2-42010af0021e
secrets:
- name: build-robot-token-bvbk5
```

*then you will see that a token has automatically been created and is referenced by the service account.*

*You may use authorization plugins to [set permissions on service accounts](#).*

*To use a non-default service account, simply set the `spec.serviceAccountName` field of a pod to the name of the service account you wish to use.*

*The service account has to exist at the time the pod is created, or it will be rejected.*

*You cannot update the service account of an already created pod.*

*You can clean up the service account from this example like this:*

```
kubectl delete serviceaccount/build-robot
```

## **Manually create a service account API token.**

*Suppose we have an existing service account named "build-robot" as mentioned above, and we create a new secret manually.*

```
kubectl apply -f - <<EOF
apiVersion: v1
kind: Secret
metadata:
  name: build-robot-secret
  annotations:
    kubernetes.io/service-account.name: build-robot
type: kubernetes.io/service-account-token
EOF
```

*Now you can confirm that the newly built secret is populated with an API token for the "build-robot" service account.*

*Any tokens for non-existent service accounts will be cleaned up by the token controller.*

```
kubectl describe secrets/build-robot-secret
```

*The output is similar to this:*

```
Name:          build-robot-secret
Namespace:     default
Labels:        <none>
Annotations:   kubernetes.io/service-account.name=build-robot
               kubernetes.io/service-
               account.uid=da68f9c6-9d26-11e7-b84e-002dc52800da
Type:         kubernetes.io/service-account-token

Data
=====
ca.crt:       1338 bytes
namespace:    7 bytes
token:        ...
```

**Note:** The content of token is elided here.

## Add **ImagePullSecrets** to a service account

### Create an **imagePullSecret**

- Create an *imagePullSecret*, as described in [Specifying ImagePullSecrets on a Pod](#).

```
kubectl create secret docker-registry myregistrykey --docker-
server=DUMMY_SERVER \
      --docker-username=DUMMY_USERNAME --docker-password=DU
      MMY_DOCKER_PASSWORD \
      --docker-email=DUMMY_DOCKER_EMAIL
```

- Verify it has been created.

```
kubectl get secrets myregistrykey
```

*The output is similar to this:*

NAME	TYPE	DATA
AGE		

```
myregistrykey    kubernetes.io/.dockerconfigjson  1  Â
Â 1d
```

## Add image pull secret to service account

Next, modify the default service account for the namespace to use this secret as an `imagePullSecret`.

```
kubectl patch serviceaccount default -p '{"imagePullSecrets": [{"name": "myregistrykey"}]}'
```

You can instead use `kubectl edit`, or manually edit the YAML manifests as shown below:

```
kubectl get serviceaccounts default -o yaml > ./sa.yaml
```

The output of the `sa.yaml` file is similar to this:

```
apiVersion: v1
kind: ServiceAccount
metadata:
  creationTimestamp: 2015-08-07T22:02:39Z
  name: default
  namespace: default
  resourceVersion: "243024"
  uid: 052fb0f4-3d50-11e5-b066-42010af0d7b6
secrets:
- name: default-token-uudge
```

Using your editor of choice (for example `vi`), open the `sa.yaml` file, delete line with key `resourceVersion`, add lines with `imagePullSecrets:` and save.

The output of the `sa.yaml` file is similar to this:

```
apiVersion: v1
kind: ServiceAccount
metadata:
  creationTimestamp: 2015-08-07T22:02:39Z
  name: default
  namespace: default
  uid: 052fb0f4-3d50-11e5-b066-42010af0d7b6
secrets:
- name: default-token-uudge
```

```
imagePullSecrets:  
- name: myregistrykey
```

Finally replace the serviceaccount with the new updated `sa.yaml` file

```
kubectl replace serviceaccount default -f ./sa.yaml
```

## Verify `imagePullSecrets` was added to pod spec

Now, when a new Pod is created in the current namespace and using the default ServiceAccount, the new Pod has its `spec.imagePullSecrets` field set automatically:

```
kubectl run nginx --image=nginx --restart=Never  
kubectl get pod nginx -o=jsonpath='{.spec.imagePullSecrets[0].name}'
```

The output is:

```
myregistrykey
```

## Service Account Token Volume Projection

**FEATURE STATE:** Kubernetes v1.20 [stable]

### Note:

To enable and use token request projection, you must specify each of the following command line arguments to `kube-apiserver`:

- `--service-account-issuer`
- `--service-account-key-file`
- `--service-account-signing-key-file`
- `--api-audiences`

The kubelet can also project a service account token into a Pod. You can specify desired properties of the token, such as the audience and the validity duration. These properties are not configurable on the default service account token. The service account token will also become invalid against the API when the Pod or the ServiceAccount is deleted.

This behavior is configured on a PodSpec using a `ProjectedVolume` type called [ServiceAccountToken](#). To provide a pod with a token with an audience

of "vault" and a validity duration of two hours, you would configure the following in your PodSpec:

[pods/pod-projected-svc-token.yaml](#)  


```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
    - image: nginx
      name: nginx
    volumeMounts:
      - mountPath: /var/run/secrets/tokens
        name: vault-token
  serviceAccountName: build-robot
  volumes:
    - name: vault-token
  projected:
    sources:
      - serviceAccountToken:
          path: vault-token
          expirationSeconds: 7200
          audience: vault
```

Create the Pod:

```
kubectl create -f https://k8s.io/examples/pods/pod-projected-svc-
token.yaml
```

The kubelet will request and store the token on behalf of the pod, make the token available to the pod at a configurable file path, and refresh the token as it approaches expiration. Kubelet proactively rotates the token if it is older than 80% of its total TTL, or if the token is older than 24 hours.

The application is responsible for reloading the token when it rotates. Periodic reloading (e.g. once every 5 minutes) is sufficient for most use cases.

## **Service Account Issuer Discovery**

**FEATURE STATE:** Kubernetes v1.20 [beta]

The Service Account Issuer Discovery feature is enabled by enabling the `ServiceAccountIssuerDiscovery` [feature gate](#) and then enabling the Service Account Token Projection feature as described [above](#).

**Note:**

The issuer URL must comply with the [OIDC Discovery Spec](#). In practice, this means it must use the `https` scheme, and should serve an OpenID provider configuration at `{service-account-issuer}/.well-known/openid-configuration`.

If the URL does not comply, the `ServiceAccountIssuerDiscovery` endpoints will not be registered, even if the feature is enabled.

The Service Account Issuer Discovery feature enables federation of Kubernetes service account tokens issued by a cluster (the identity provider) with external systems (relying parties).

When enabled, the Kubernetes API server provides an OpenID Provider Configuration document at `/.well-known/openid-configuration` and the associated JSON Web Key Set (JWKS) at `/openid/v1/jwks`. The OpenID Provider Configuration is sometimes referred to as the discovery document.

When enabled, the cluster is also configured with a default RBAC `ClusterRole` called `system:service-account-issuer-discovery`. No role bindings are provided by default. Administrators may, for example, choose whether to bind the role to `system:authenticated` or `system:unauthenticated` depending on their security requirements and which external systems they intend to federate with.

**Note:** The responses served at `/.well-known/openid-configuration` and `/openid/v1/jwks` are designed to be OIDC compatible, but not strictly OIDC compliant. Those documents contain only the parameters necessary to perform validation of Kubernetes service account tokens.

The JWKS response contains public keys that a relying party can use to validate the Kubernetes service account tokens. Relying parties first query for the OpenID Provider Configuration, and use the `jwks_uri` field in the response to find the JWKS.

In many cases, Kubernetes API servers are not available on the public internet, but public endpoints that serve cached responses from the API server can be made available by users or service providers. In these cases, it is possible to override the `jwks_uri` in the OpenID Provider Configuration so that it points to the public endpoint, rather than the API server's address, by

passing the `--service-account-jwks-uri` flag to the API server. Like the issuer URL, the JWKS URI is required to use the `https` scheme.

## What's next

See also:

- [Cluster Admin Guide to Service Accounts](#)
- [Service Account Signing Key Retrieval KEP](#)
- [OIDC Discovery Spec](#)

## 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](#).

Last modified November 04, 2020 at 11:24 AM PST: [TokenRequest and TokenRequestProjection are GA now \(#24823\) \(3590d7300\)](#)  
[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Use the Default Service Account to access the API server](#).
- [Use Multiple Service Accounts](#).
- [Manually create a service account API token](#).
- [Add ImagePullSecrets to a service account](#)
  - [Create an imagePullSecret](#)
  - [Add image pull secret to service account](#)
  - [Verify imagePullSecrets was added to pod spec](#)
- [Service Account Token Volume Projection](#)
- [Service Account Issuer Discovery](#)
- [What's next](#)

# Pull an Image from a Private Registry

This page shows how to create a Pod that uses a Secret to pull an image from a private Docker registry or repository.

## Before you begin

-

You need to have a Kubernetes cluster, and the `kubectl` command-line tool must be configured to communicate with your cluster. If you do not already have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:

- [Katacoda](#)
- [Play with Kubernetes](#)

To check the version, enter `kubectl version`.

- To do this exercise, you need a [Docker ID](#) and password.

## Log in to Docker

On your laptop, you must authenticate with a registry in order to pull a private image:

```
docker login
```

When prompted, enter your Docker username and password.

The login process creates or updates a `config.json` file that holds an authorization token.

View the `config.json` file:

```
cat ~/.docker/config.json
```

The output contains a section similar to this:

```
{
  "auths": {
    "https://index.docker.io/v1/": {
      "auth": "c3R...zE2"
    }
  }
}
```

**Note:** If you use a Docker credentials store, you won't see that `auth` entry but a `credsStore` entry with the name of the store as value.

## Create a Secret based on existing Docker credentials

A Kubernetes cluster uses the `Secret` of `docker-registry` type to authenticate with a container registry to pull a private image.

If you already ran `docker login`, you can copy that credential into Kubernetes:

```
kubectl create secret generic regcred \
  --from-file=.dockerconfigjson=<path/to/.docker/config.json> \
  --type=kubernetes.io/dockerconfigjson
```

If you need more control (for example, to set a namespace or a label on the new secret) then you can customise the Secret before storing it. Be sure to:

- set the name of the data item to `.dockerconfigjson`
- base64 encode the docker file and paste that string, unbroken as the value for field `data[".dockerconfigjson"]`
- set type to `kubernetes.io/dockerconfigjson`

Example:

```
apiVersion: v1
kind: Secret
metadata:
  name: myregistrykey
  namespace: awesomeapps
data:
  .dockerconfigjson: UmVhbGx5IHJ1YWxseSByZWVlZWVlZWVlZWFnYWFnYWFn...
  type: kubernetes.io/dockerconfigjson
```

If you get the error message `error: no objects passed to create`, it may mean the base64 encoded string is invalid. If you get an error message like `Secret "myregistrykey" is invalid: data[".dockerconfigjson": invalid value ...]`, it means the base64 encoded string in the data was successfully decoded, but could not be parsed as a `.docker/config.json` file.

## Create a Secret by providing credentials on the command line

Create this Secret, naming it `regcred`:

```
kubectl create secret docker-registry regcred --docker-server=<your-registry-server> --docker-username=<your-name> --docker-password=<your-pword> --docker-email=<your-email>
```

where:

- `<your-registry-server>` is your Private Docker Registry FQDN. (<https://index.docker.io/v1/> for DockerHub)
- `<your-name>` is your Docker username.
- `<your-pword>` is your Docker password.
- `<your-email>` is your Docker email.

You have successfully set your Docker credentials in the cluster as a Secret called `regcred`.

**Note:** Typing secrets on the command line may store them in your shell history unprotected, and those secrets might also be visible to other users on your PC during the time that `kubectl` is running.

## Inspecting the Secret `regcred`

To understand the contents of the `regcred` Secret you just created, start by viewing the Secret in YAML format:

```
kubectl get secret regcred --output=yaml
```

The output is similar to this:

```
apiVersion: v1
kind: Secret
metadata:
  ...
  name: regcred
  ...
data:
  .dockerconfigjson: eyJodHRwczovL2luZGV4L ... J0QUL6RTIifX0=
type: kubernetes.io/dockerconfigjson
```

The value of the `.dockerconfigjson` field is a base64 representation of your Docker credentials.

To understand what is in the `.dockerconfigjson` field, convert the secret data to a readable format:

```
kubectl get secret regcred --output="jsonpath={.data.\.dockerconfigjson}" | base64 --decode
```

The output is similar to this:

```
{"auths": {"your.private.registry.example.com": {"username": "janedoe", "password": "xxxxxxxxxxxx", "email": "jdoe@example.com", "auth": "c3R...zE2"}}}
```

To understand what is in the `auth` field, convert the base64-encoded data to a readable format:

```
echo "c3R...zE2" | base64 --decode
```

The output, `username` and `password` concatenated with a `:`, is similar to this:

```
janedoe:xxxxxxxxxxxx
```

Notice that the Secret data contains the authorization token similar to your local `~/.docker/config.json` file.

You have successfully set your Docker credentials as a Secret called `regcred` in the cluster.

## Create a Pod that uses your Secret

Here is a configuration file for a Pod that needs access to your Docker credentials in `regcred`:

[`pods/private-reg-pod.yaml`](#)



```
apiVersion: v1
kind: Pod
metadata:
  name: private-reg
spec:
  containers:
    - name: private-reg-container
      image: <your-private-image>
  imagePullSecrets:
    - name: regcred
```

Download the above file:

```
wget -O my-private-reg-pod.yaml https://k8s.io/examples/pods/
private-reg-pod.yaml
```

In file `my-private-reg-pod.yaml`, replace `<your-private-image>` with the path to an image in a private registry such as:

```
your.private.registry.example.com/janedoe/jdoe-private:v1
```

To pull the image from the private registry, Kubernetes needs credentials. The `imagePullSecrets` field in the configuration file specifies that Kubernetes should get the credentials from a Secret named `regcred`.

Create a Pod that uses your Secret, and verify that the Pod is running:

```
kubectl apply -f my-private-reg-pod.yaml
kubectl get pod private-reg
```

## What's next

- Learn more about [Secrets](#).
- Learn more about [using a private registry](#).
- Learn more about [adding image pull secrets to a service account](#).
- See [kubectl create secret docker-registry](#).
- See [Secret](#).
- See the `imagePullSecrets` field of [PodSpec](#).

# 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](#).

Last modified May 30, 2020 at 3:10 PM PST: [add en pages \(ecc27bbbe\)](#)  
[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Log in to Docker](#)
- [Create a Secret based on existing Docker credentials](#)
- [Create a Secret by providing credentials on the command line](#)
- [Inspecting the Secret regcred](#)
- [Create a Pod that uses your Secret](#)
- [What's next](#)

# Configure Liveness, Readiness and Startup Probes

This page shows how to configure liveness, readiness and startup probes for containers.

The [kubelet](#) uses liveness probes to know when to restart a container. For example, liveness probes could catch a deadlock, where an application is running, but unable to make progress. Restarting a container in such a state can help to make the application more available despite bugs.

The kubelet uses readiness probes to know when a container is ready to start accepting traffic. A Pod is considered ready when all of its containers are ready. One use of this signal is to control which Pods are used as backends for Services. When a Pod is not ready, it is removed from Service load balancers.

The kubelet uses startup probes to know when a container application has started. If such a probe is configured, it disables liveness and readiness checks until it succeeds, making sure those probes don't interfere with the application startup. This can be used to adopt liveness checks on slow starting containers, avoiding them getting killed by the kubelet before they are up and running.

## Before you begin

You need to have a Kubernetes cluster, and the `kubectl` command-line tool must be configured to communicate with your cluster. If you do not already

have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:

- [Katacoda](#)
- [Play with Kubernetes](#)

To check the version, enter `kubectl version`.

## Define a liveness command

Many applications running for long periods of time eventually transition to broken states, and cannot recover except by being restarted. Kubernetes provides liveness probes to detect and remedy such situations.

In this exercise, you create a Pod that runs a container based on the `k8s.gcr.io/busybox` image. Here is the configuration file for the Pod:

[pods/probe/exec-liveness.yaml](#)



```
apiVersion: v1
kind: Pod
metadata:
  labels:
    test: liveness
    name: liveness-exec
spec:
  containers:
    - name: liveness
      image: k8s.gcr.io/busybox
      args:
        - /bin/sh
        - -c
        - touch /tmp/healthy; sleep 30; rm -rf /tmp/healthy; sleep 60
  livenessProbe:
    exec:
      command:
        - cat
        - /tmp/healthy
    initialDelaySeconds: 5
    periodSeconds: 5
```

In the configuration file, you can see that the Pod has a single Container. The `periodSeconds` field specifies that the kubelet should perform a liveness probe every 5 seconds. The `initialDelaySeconds` field tells the kubelet that it should wait 5 seconds before performing the first probe. To perform a probe, the kubelet executes the command `cat /tmp/healthy` in the target container. If the command succeeds, it returns 0, and the kubelet considers

*the container to be alive and healthy. If the command returns a non-zero value, the kubelet kills the container and restarts it.*

*When the container starts, it executes this command:*

```
/bin/sh -c "touch /tmp/healthy; sleep 30; rm -rf /tmp/healthy;  
sleep 600"
```

*For the first 30 seconds of the container's life, there is a /tmp/healthy file. So during the first 30 seconds, the command cat /tmp/healthy returns a success code. After 30 seconds, cat /tmp/healthy returns a failure code.*

*Create the Pod:*

```
kubectl apply -f https://k8s.io/examples/pods/probe/exec-liveness.yaml
```

*Within 30 seconds, view the Pod events:*

```
kubectl describe pod liveness-exec
```

*The output indicates that no liveness probes have failed yet:*

FirstSeen SubobjectPath	LastSeen	Count	From	Type	Reason	Message
24s	24s	1	{default-scheduler }			
Normal	Scheduled					Successfully assigned liveness-exec to worker0
23s	23s	1	{kubelet worker0}	spec.containers{liveness}	Normal	Pulling pulling image "k8s.gcr.io/busybox"
23s	23s	1	{kubelet worker0}	spec.containers{liveness}	Normal	Pulled Successfully pulled image "k8s.gcr.io/busybox"
23s	23s	1	{kubelet worker0}	spec.containers{liveness}	Normal	Created Created container with docker id 86849c15382e; Security: [seccomp=unconfined]
23s	23s	1	{kubelet worker0}	spec.containers{liveness}	Normal	Started Started container with docker id 86849c15382e

*After 35 seconds, view the Pod events again:*

```
kubectl describe pod liveness-exec
```

At the bottom of the output, there are messages indicating that the liveness probes have failed, and the containers have been killed and recreated.

FirstSeen	LastSeen	Count	From	Type	Reason	Message
37s	37s	1	{default-scheduler }			
Normal	Scheduled					Successfully assigned liveness-exec to worker0
36s	36s	1	{kubelet worker0}			
spec.containers{liveness}	Normal		Pulling			pulling image "k8s.gcr.io/busybox"
36s	36s	1	{kubelet worker0}			
spec.containers{liveness}	Normal		Pulled			Successfully pulled image "k8s.gcr.io/busybox"
36s	36s	1	{kubelet worker0}			
spec.containers{liveness}	Normal		Created			Created container with docker id 86849c15382e; Security: [seccomp=unconfined]
36s	36s	1	{kubelet worker0}			
spec.containers{liveness}	Normal		Started			Started container with docker id 86849c15382e
2s	2s	1	{kubelet worker0}			
spec.containers{liveness}	Warning		Unhealthy			Liveness probe failed: cat: can't open '/tmp/healthy': No such file or directory

Wait another 30 seconds, and verify that the container has been restarted:

```
kubectl get pod liveness-exec
```

The output shows that RESTARTS has been incremented:

NAME	READY	STATUS	RESTARTS	AGE
liveness-exec	1/1	Running	1	1m

## Define a liveness HTTP request

Another kind of liveness probe uses an HTTP GET request. Here is the configuration file for a Pod that runs a container based on the k8s.gcr.io/liveness image.



```
apiVersion: v1
kind: Pod
metadata:
  labels:
    test: liveness
  name: liveness-http
spec:
  containers:
    - name: liveness
      image: k8s.gcr.io/liveness
      args:
        - /server
      livenessProbe:
        httpGet:
          path: /healthz
          port: 8080
          httpHeaders:
            - name: Custom-Header
              value: Awesome
      initialDelaySeconds: 3
      periodSeconds: 3
```

In the configuration file, you can see that the Pod has a single container. The `periodSeconds` field specifies that the kubelet should perform a liveness probe every 3 seconds. The `initialDelaySeconds` field tells the kubelet that it should wait 3 seconds before performing the first probe. To perform a probe, the kubelet sends an HTTP GET request to the server that is running in the container and listening on port 8080. If the handler for the server's `/healthz` path returns a success code, the kubelet considers the container to be alive and healthy. If the handler returns a failure code, the kubelet kills the container and restarts it.

Any code greater than or equal to 200 and less than 400 indicates success. Any other code indicates failure.

You can see the source code for the server in [server.go](#).

For the first 10 seconds that the container is alive, the `/healthz` handler returns a status of 200. After that, the handler returns a status of 500.

```
http.HandleFunc("/healthz", func(w http.ResponseWriter, r *http.Request) {
    duration := time.Now().Sub(started)
    if duration.Seconds() > 10 {
        w.WriteHeader(500)
```

```
w.Write([]byte(fmt.Sprintf("error: %v", duration.Seconds())))
})} else {
    w.WriteHeader(200)
    w.Write([]byte("ok"))
}
})
```

The kubelet starts performing health checks 3 seconds after the container starts. So the first couple of health checks will succeed. But after 10 seconds, the health checks will fail, and the kubelet will kill and restart the container.

To try the HTTP liveness check, create a Pod:

```
kubectl apply -f https://k8s.io/examples/pods/probe/http-liveness.yaml
```

After 10 seconds, view Pod events to verify that liveness probes have failed and the container has been restarted:

```
kubectl describe pod liveness-http
```

In releases prior to v1.13 (including v1.13), if the environment variable `http_proxy` (or `HTTP_PROXY`) is set on the node where a Pod is running, the HTTP liveness probe uses that proxy. In releases after v1.13, local HTTP proxy environment variable settings do not affect the HTTP liveness probe.

## Define a TCP liveness probe

A third type of liveness probe uses a TCP socket. With this configuration, the kubelet will attempt to open a socket to your container on the specified port. If it can establish a connection, the container is considered healthy, if it can't it is considered a failure.

[pods/probe/tcp-liveness-readiness.yaml](#)  


```
apiVersion: v1
kind: Pod
metadata:
  name: goproxy
  labels:
    app: goproxy
spec:
```

```
containers:
- name: goproxy
  image: k8s.gcr.io/goproxy:0.1
  ports:
  - containerPort: 8080
    readinessProbe:
      tcpSocket:
        port: 8080
      initialDelaySeconds: 5
      periodSeconds: 10
    livenessProbe:
      tcpSocket:
        port: 8080
      initialDelaySeconds: 15
      periodSeconds: 20
```

As you can see, configuration for a TCP check is quite similar to an HTTP check. This example uses both readiness and liveness probes. The kubelet will send the first readiness probe 5 seconds after the container starts. This will attempt to connect to the goproxy container on port 8080. If the probe succeeds, the Pod will be marked as ready. The kubelet will continue to run this check every 10 seconds.

In addition to the readiness probe, this configuration includes a liveness probe. The kubelet will run the first liveness probe 15 seconds after the container starts. Just like the readiness probe, this will attempt to connect to the goproxy container on port 8080. If the liveness probe fails, the container will be restarted.

To try the TCP liveness check, create a Pod:

```
kubectl apply -f https://k8s.io/examples/pods/probe/tcp-liveness-readiness.yaml
```

After 15 seconds, view Pod events to verify that liveness probes:

```
kubectl describe pod goproxy
```

## Use a named port

You can use a named [ContainerPort](#) for HTTP or TCP liveness checks:

```
ports:
- name: liveness-port
```

```
  containerPort: 8080
  hostPort: 8080

  livenessProbe:
    httpGet:
      path: /healthz
      port: liveness-port
```

## **Protect slow starting containers with startup probes**

*Sometimes, you have to deal with legacy applications that might require an additional startup time on their first initialization. In such cases, it can be tricky to set up liveness probe parameters without compromising the fast response to deadlocks that motivated such a probe. The trick is to set up a startup probe with the same command, HTTP or TCP check, with a failureThreshold \* periodSeconds long enough to cover the worse case startup time.*

*So, the previous example would become:*

```
  ports:
  - name: liveness-port
    containerPort: 8080
    hostPort: 8080

  livenessProbe:
    httpGet:
      path: /healthz
      port: liveness-port
    failureThreshold: 1
    periodSeconds: 10

  startupProbe:
    httpGet:
      path: /healthz
      port: liveness-port
    failureThreshold: 30
    periodSeconds: 10
```

*Thanks to the startup probe, the application will have a maximum of 5 minutes ( $30 * 10 = 300s$ ) to finish its startup. Once the startup probe has succeeded once, the liveness probe takes over to provide a fast response to container deadlocks. If the startup probe never succeeds, the container is killed after 300s and subject to the pod's restartPolicy.*

## Define readiness probes

Sometimes, applications are temporarily unable to serve traffic. For example, an application might need to load large data or configuration files during startup, or depend on external services after startup. In such cases, you don't want to kill the application, but you don't want to send it requests either. Kubernetes provides readiness probes to detect and mitigate these situations. A pod with containers reporting that they are not ready does not receive traffic through Kubernetes Services.

**Note:** Readiness probes runs on the container during its whole lifecycle.

Readiness probes are configured similarly to liveness probes. The only difference is that you use the `readinessProbe` field instead of the `livenessProbe` field.

```
readinessProbe:  
  exec:  
    command:  
    - cat  
    - /tmp/healthy  
  initialDelaySeconds: 5  
  periodSeconds: 5
```

Configuration for HTTP and TCP readiness probes also remains identical to liveness probes.

Readiness and liveness probes can be used in parallel for the same container. Using both can ensure that traffic does not reach a container that is not ready for it, and that containers are restarted when they fail.

## Configure Probes

[Probes](#) have a number of fields that you can use to more precisely control the behavior of liveness and readiness checks:

- `initialDelaySeconds`: Number of seconds after the container has started before liveness or readiness probes are initiated. Defaults to 0 seconds. Minimum value is 0.
- `periodSeconds`: How often (in seconds) to perform the probe. Default to 10 seconds. Minimum value is 1.
- `timeoutSeconds`: Number of seconds after which the probe times out. Defaults to 1 second. Minimum value is 1.

- *successThreshold*: Minimum consecutive successes for the probe to be considered successful after having failed. Defaults to 1. Must be 1 for liveness and startup Probes. Minimum value is 1.
- *failureThreshold*: When a probe fails, Kubernetes will try *failureThreshold* times before giving up. Giving up in case of liveness probe means restarting the container. In case of readiness probe the Pod will be marked Unready. Defaults to 3. Minimum value is 1.

**Note:**

*Before Kubernetes 1.20, the field `timeoutSeconds` was not respected for exec probes: probes continued running indefinitely, even past their configured deadline, until a result was returned.*

*This defect was corrected in Kubernetes v1.20. You may have been relying on the previous behavior, even without realizing it, as the default timeout is 1 second. As a cluster administrator, you can disable the [feature gate](#) `ExecProbeTimeout` (set it to `false`) on each kubelet to restore the behavior from older versions, then remove that override once all the exec probes in the cluster have a `timeoutSeconds` value set.*

*If you have pods that are impacted from the default 1 second timeout, you should update their probe timeout so that you're ready for the eventual removal of that feature gate.*

*With the fix of the defect, for exec probes, on Kubernetes 1.20+ with the `dockershim` container runtime, the process inside the container may keep running even after probe returned failure because of the timeout.*

**Caution:** Incorrect implementation of readiness probes may result in an ever growing number of processes in the container, and resource starvation if this is left unchecked.

## HTTP probes

[HTTP probes](#) have additional fields that can be set on `httpGet`:

- *host*: Host name to connect to, defaults to the pod IP. You probably want to set "Host" in `httpHeaders` instead.
- *scheme*: Scheme to use for connecting to the host (HTTP or HTTPS). Defaults to HTTP.
- *path*: Path to access on the HTTP server.
- *httpHeaders*: Custom headers to set in the request. HTTP allows repeated headers.
- *port*: Name or number of the port to access on the container. Number must be in the range 1 to 65535.

*For an HTTP probe, the kubelet sends an HTTP request to the specified path and port to perform the check. The kubelet sends the probe to the pod's IP address, unless the address is overridden by the optional host field in httpGet. If scheme field is set to HTTPS, the kubelet sends an HTTPS request skipping the certificate verification. In most scenarios, you do not want to set the host field. Here's one scenario where you would set it. Suppose the container listens on 127.0.0.1 and the Pod's hostNetwork field is true. Then host, under httpGet, should be set to 127.0.0.1. If your pod relies on virtual hosts, which is probably the more common case, you should not use host, but rather set the Host header in httpHeaders.*

*For an HTTP probe, the kubelet sends two request headers in addition to the mandatory Host header: User-Agent, and Accept. The default values for these headers are kube-probe/1.20 (where 1.20 is the version of the kubelet ), and \*/\* respectively.*

*You can override the default headers by defining .httpHeaders for the probe; for example*

```
livenessProbe:  
  httpHeaders:  
    Accept: application/json  
  
startupProbe:  
  httpHeaders:  
    User-Agent: MyUserAgent
```

*You can also remove these two headers by defining them with an empty value.*

```
livenessProbe:  
  httpHeaders:  
    Accept: ""  
  
startupProbe:  
  httpHeaders:  
    User-Agent: ""
```

## **TCP probes**

*For a TCP probe, the kubelet makes the probe connection at the node, not in the pod, which means that you can not use a service name in the host parameter since the kubelet is unable to resolve it.*

## **What's next**

- Learn more about [Container Probes](#).

You can also read the API references for:

- [Pod](#)
- [Container](#)
- [Probe](#)

## **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](#).

Last modified November 25, 2020 at 5:29 PM PST: [Updating docs/tasks/configure-pod-container/configure-liveness-readiness-startup-probes \(f205837b9\)](#)

[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Define a liveness command](#)
- [Define a liveness HTTP request](#)
- [Define a TCP liveness probe](#)
- [Use a named port](#)
- [Protect slow starting containers with startup probes](#)
- [Define readiness probes](#)
- [Configure Probes](#)
  - [HTTP probes](#)
  - [TCP probes](#)
- [What's next](#)

## **Assign Pods to Nodes**

This page shows how to assign a Kubernetes Pod to a particular node in a Kubernetes cluster.

### **Before you begin**

You need to have a Kubernetes cluster, and the `kubectl` command-line tool must be configured to communicate with your cluster. If you do not already have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:

- [Katacoda](#)
- [Play with Kubernetes](#)

To check the version, enter `kubectl version`.

## Add a label to a node

1. List the [nodes](#) in your cluster, along with their labels:

```
kubectl get nodes --show-labels
```

The output is similar to this:

NAME	STATUS	ROLES	AGE	VERSION	LABELS
worker0	Ready	<none>	1d	v1.13.0	..., kubernetes.io/hostname=worker0
worker1	Ready	<none>	1d	v1.13.0	..., kubernetes.io/hostname=worker1
worker2	Ready	<none>	1d	v1.13.0	..., kubernetes.io/hostname=worker2

2. Choose one of your nodes, and add a label to it:

```
kubectl label nodes <your-node-name> disktype:ssd
```

where `<your-node-name>` is the name of your chosen node.

3. Verify that your chosen node has a `disktype:ssd` label:

```
kubectl get nodes --show-labels
```

The output is similar to this:

NAME	STATUS	ROLES	AGE	VERSION	LABELS
worker0	Ready	<none>	1d	v1.13.0	..., disktype:ssd, kubernetes.io/hostname=worker0
worker1	Ready	<none>	1d	v1.13.0	..., kubernetes.io/hostname=worker1
worker2	Ready	<none>	1d	v1.13.0	..., kubernetes.io/hostname=worker2

In the preceding output, you can see that the `worker0` node has a `disktype:ssd` label.

## **Create a pod that gets scheduled to your chosen node**

This pod configuration file describes a pod that has a node selector, `disktype: ssd`. This means that the pod will get scheduled on a node that has a `disktype=ssd` label.

[pods/pod-nginx.yaml](#)



```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    env: test
spec:
  containers:
  - name: nginx
    image: nginx
    imagePullPolicy: IfNotPresent
  nodeSelector:
    disktype: ssd
```

1. Use the configuration file to create a pod that will get scheduled on your chosen node:

```
kubectl apply -f https://k8s.io/examples/pods/pod-nginx.yaml
```

2. Verify that the pod is running on your chosen node:

```
kubectl get pods --output=wide
```

The output is similar to this:

NAME	READY	STATUS	RESTARTS	AGE	IP
nginx	1/1	Running	0	13s	10.200.0.4
worker0					

## **Create a pod that gets scheduled to specific node**

You can also schedule a pod to one specific node via setting `nodeName`.



```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  nodeName: foo-node # schedule pod to specific node
  containers:
    - name: nginx
      image: nginx
      imagePullPolicy: IfNotPresent
```

Use the configuration file to create a pod that will get scheduled on `foo-node` only.

## What's next

- Learn more about [labels and selectors](#).
- Learn more about [nodes](#).

## 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](#).

Last modified May 30, 2020 at 3:10 PM PST: [add en pages \(ecc27bbbe\)](#)  
[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Add a label to a node](#)
- [Create a pod that gets scheduled to your chosen node](#)
- [Create a pod that gets scheduled to specific node](#)
- [What's next](#)

# Assign Pods to Nodes using Node Affinity

This page shows how to assign a Kubernetes Pod to a particular node using Node Affinity in a Kubernetes cluster.

## Before you begin

You need to have a Kubernetes cluster, and the `kubectl` command-line tool must be configured to communicate with your cluster. If you do not already have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:

- [Katacoda](#)
- [Play with Kubernetes](#)

Your Kubernetes server must be at or later than version v1.10. To check the version, enter `kubectl version`.

## Add a label to a node

1. List the nodes in your cluster, along with their labels:

```
kubectl get nodes --show-labels
```

The output is similar to this:

NAME	STATUS	ROLES	AGE	VERSION	LABELS
worker0	Ready	<none>	1d	v1.13.0	..., kubernetes.io/hostname=worker0
worker1	Ready	<none>	1d	v1.13.0	..., kubernetes.io/hostname=worker1
worker2	Ready	<none>	1d	v1.13.0	..., kubernetes.io/hostname=worker2

2. Choose one of your nodes, and add a label to it:

```
kubectl label nodes <your-node-name> disktype:ssd
```

where `<your-node-name>` is the name of your chosen node.

3. Verify that your chosen node has a `disktype:ssd` label:

```
kubectl get nodes --show-labels
```

The output is similar to this:

NAME	STATUS	ROLES	AGE	VERSION	LABELS
worker0	Ready	<none>	1d		

```
v1.13.0      ... , disktype=ssd, kubernetes.io/
hostname=worker0
worker1  Ready     <none>    1d
v1.13.0      ... , kubernetes.io/hostname=worker1
worker2  Ready     <none>    1d
v1.13.0      ... , kubernetes.io/hostname=worker2
```

In the preceding output, you can see that the `worker0` node has a `disktype=ssd` label.

## Schedule a Pod using required node affinity

This manifest describes a Pod that has a `requiredDuringSchedulingIgnoredDuringExecution` node affinity, `disktype: ssd`. This means that the pod will get scheduled only on a node that has a `disktype=ssd` label.

[pods/pod-nginx-required-affinity.yaml](#)  


```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: disktype
            operator: In
            values:
            - ssd
  containers:
  - name: nginx
    image: nginx
    imagePullPolicy: IfNotPresent
```

1. Apply the manifest to create a Pod that is scheduled onto your chosen node:

```
kubectl apply -f https://k8s.io/examples/pods/pod-nginx-
required-affinity.yaml
```

2. Verify that the pod is running on your chosen node:

```
kubectl get pods --output=wide
```

The output is similar to this:

NAME	READY	STATUS	RESTARTS	AGE	IP
NODE					
nginx	1/1	Running	0	13s	10.200.0.4
worker0					

## Schedule a Pod using preferred node affinity

This manifest describes a Pod that has a `preferredDuringSchedulingIgnoredDuringExecution` node affinity, `disktype: ssd`. This means that the pod will prefer a node that has a `disktype=ssd` label.

[pods/pod-nginx-preferred-affinity.yaml](#)

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  affinity:
    nodeAffinity:
      preferredDuringSchedulingIgnoredDuringExecution:
        - weight: 1
          preference:
            matchExpressions:
              - key: disktype
                operator: In
                values:
                  - ssd
  containers:
    - name: nginx
      image: nginx
      imagePullPolicy: IfNotPresent
```

1. Apply the manifest to create a Pod that is scheduled onto your chosen node:

```
kubectl apply -f https://k8s.io/examples/pods/pod-nginx-preferred-affinity.yaml
```

2. Verify that the pod is running on your chosen node:

```
kubectl get pods --output=wide
```

The output is similar to this:

NAME	READY	STATUS	RESTARTS	AGE	IP
NODE					
nginx	1/1	Running	0	13s	10.200.0.4
worker0					

nginx	1/1	Running	0	13s	10.200.0.4
worker0					

## What's next

Learn more about [Node Affinity](#).

## 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](#).

Last modified May 30, 2020 at 3:10 PM PST: [add en pages \(ecc27bbbe\)](#)  
[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Add a label to a node](#)
- [Schedule a Pod using required node affinity](#)
- [Schedule a Pod using preferred node affinity](#)
- [What's next](#)

# Configure Pod Initialization

This page shows how to use an Init Container to initialize a Pod before an application Container runs.

## Before you begin

You need to have a Kubernetes cluster, and the `kubectl` command-line tool must be configured to communicate with your cluster. If you do not already have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:

- [Katacoda](#)
- [Play with Kubernetes](#)

To check the version, enter `kubectl version`.

## Create a Pod that has an Init Container

*In this exercise you create a Pod that has one application Container and one Init Container. The init container runs to completion before the application container starts.*

*Here is the configuration file for the Pod:*

[pods/init-containers.yaml](#)



```
apiVersion: v1
kind: Pod
metadata:
  name: init-demo
spec:
  containers:
    - name: nginx
      image: nginx
      ports:
        - containerPort: 80
      volumeMounts:
        - name: workdir
          mountPath: /usr/share/nginx/html
    # These containers are run during pod initialization
    initContainers:
      - name: install
        image: busybox
        command:
          - wget
          - "-O"
          - "/work-dir/index.html"
          - http://info.cern.ch
        volumeMounts:
          - name: workdir
            mountPath: "/work-dir"
  dnsPolicy: Default
  volumes:
    - name: workdir
      emptyDir: {}
```

*In the configuration file, you can see that the Pod has a Volume that the init container and the application container share.*

*The init container mounts the shared Volume at /work-dir, and the application container mounts the shared Volume at /usr/share/nginx/html. The init container runs the following command and then terminates:*

```
wget -O /work-dir/index.html http://info.cern.ch
```

*Notice that the init container writes the `index.html` file in the root directory of the nginx server.*

*Create the Pod:*

```
kubectl apply -f https://k8s.io/examples/pods/init-containers.yaml
```

*Verify that the nginx container is running:*

```
kubectl get pod init-demo
```

*The output shows that the nginx container is running:*

NAME	READY	STATUS	RESTARTS	AGE
init-demo	1/1	Running	0	1m

*Get a shell into the nginx container running in the init-demo Pod:*

```
kubectl exec -it init-demo -- /bin/bash
```

*In your shell, send a GET request to the nginx server:*

```
root@nginx:~# apt-get update
root@nginx:~# apt-get install curl
root@nginx:~# curl localhost
```

*The output shows that nginx is serving the web page that was written by the init container:*

```
<html><head></head><body><header>
<title>http://info.cern.ch</title>
</header>

<h1>http://info.cern.ch - home of the first website</h1>
...
<li><a href="http://info.cern.ch/hypertext/WWW/TheProject.html">Browse the first website</a></li>
...
```

## What's next

- Learn more about [communicating between Containers running in the same Pod](#).
- Learn more about [Init Containers](#).
- Learn more about [Volumes](#).
- Learn more about [Debugging Init Containers](#)

## 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](#).

Last modified October 15, 2020 at 1:26 PM PST: [Replaced URL \(0cef76025\)](#)  
[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Create a Pod that has an Init Container](#)
- [What's next](#)

# Attach Handlers to Container Lifecycle Events

This page shows how to attach handlers to Container lifecycle events. Kubernetes supports the `postStart` and `preStop` events. Kubernetes sends the `postStart` event immediately after a Container is started, and it sends the `preStop` event immediately before the Container is terminated. A Container may specify one handler per event.

## Before you begin

You need to have a Kubernetes cluster, and the `kubectl` command-line tool must be configured to communicate with your cluster. If you do not already have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:

- [Katacoda](#)
- [Play with Kubernetes](#)

To check the version, enter `kubectl version`.

## Define postStart and preStop handlers

In this exercise, you create a Pod that has one Container. The Container has handlers for the postStart and preStop events.

Here is the configuration file for the Pod:

[pods/lifecycle-events.yaml](#)



```
apiVersion: v1
kind: Pod
metadata:
  name: lifecycle-demo
spec:
  containers:
    - name: lifecycle-demo-container
      image: nginx
      lifecycle:
        postStart:
          exec:
            command: ["/bin/sh", "-c", "echo Hello from the
postStart handler > /usr/share/message"]
        preStop:
          exec:
            command: ["/bin/sh", "-c", "nginx -s quit; while killall
-0 nginx; do sleep 1; done"]
```

In the configuration file, you can see that the postStart command writes a message file to the Container's /usr/share directory. The preStop command shuts down nginx gracefully. This is helpful if the Container is being terminated because of a failure.

Create the Pod:

```
kubectl apply -f https://k8s.io/examples/pods/lifecycle-
events.yaml
```

Verify that the Container in the Pod is running:

```
kubectl get pod lifecycle-demo
```

Get a shell into the Container running in your Pod:

```
kubectl exec -it lifecycle-demo -- /bin/bash
```

In your shell, verify that the `postStart` handler created the message file:

```
root@lifecycle-demo:/# cat /usr/share/message
```

The output shows the text written by the `postStart` handler:

```
Hello from the postStart handler
```

## Discussion

Kubernetes sends the `postStart` event immediately after the Container is created. There is no guarantee, however, that the `postStart` handler is called before the Container's entrypoint is called. The `postStart` handler runs asynchronously relative to the Container's code, but Kubernetes' management of the container blocks until the `postStart` handler completes. The Container's status is not set to `RUNNING` until the `postStart` handler completes.

Kubernetes sends the `preStop` event immediately before the Container is terminated. Kubernetes' management of the Container blocks until the `preStop` handler completes, unless the Pod's grace period expires. For more details, see [Pod Lifecycle](#).

**Note:** Kubernetes only sends the `preStop` event when a Pod is terminated. This means that the `preStop` hook is not invoked when the Pod is completed. This limitation is tracked in [issue #55087](#).

## What's next

- Learn more about [Container lifecycle hooks](#).
- Learn more about the [lifecycle of a Pod](#).

## Reference

- [Lifecycle](#)
- [Container](#)
- See `terminationGracePeriodSeconds` in [PodSpec](#)

# 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](#).

Last modified December 10, 2020 at 4:27 PM PST: [State that no more than one handler is allowed per lifecycle event \(e436cf704\)](#)

[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Define postStart and preStop handlers](#)
- [Discussion](#)
- [What's next](#)
  - [Reference](#)

# Configure a Pod to Use a ConfigMap

ConfigMaps allow you to decouple configuration artifacts from image content to keep containerized applications portable. This page provides a series of usage examples demonstrating how to create ConfigMaps and configure Pods using data stored in ConfigMaps.

## Before you begin

You need to have a Kubernetes cluster, and the `kubectl` command-line tool must be configured to communicate with your cluster. If you do not already have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:

- [Katacoda](#)
- [Play with Kubernetes](#)

To check the version, enter `kubectl version`.

## Create a ConfigMap

You can use either `kubectl create configmap` or a ConfigMap generator in `kustomization.yaml` to create a ConfigMap. Note that `kubectl` starts to support `kustomization.yaml` since 1.14.

## Create a ConfigMap Using `kubectl create configmap`

Use the `kubectl create configmap` command to create ConfigMaps from [directories](#), [files](#), or [literal values](#):

```
kubectl create configmap <map-name> <data-source>
```

where `<map-name>` is the name you want to assign to the ConfigMap and `<data-source>` is the directory, file, or literal value to draw the data from. The name of a ConfigMap object must be a valid [DNS subdomain name](#).

When you are creating a ConfigMap based on a file, the key in the `<data-source>` defaults to the basename of the file, and the value defaults to the file content.

You can use [`kubectl describe`](#) or [`kubectl get`](#) to retrieve information about a ConfigMap.

### Create ConfigMaps from directories

You can use `kubectl create configmap` to create a ConfigMap from multiple files in the same directory. When you are creating a ConfigMap based on a directory, kubectl identifies files whose basename is a valid key in the directory and packages each of those files into the new ConfigMap. Any directory entries except regular files are ignored (e.g. subdirectories, symlinks, devices, pipes, etc).

For example:

```
# Create the local directory
mkdir -p configure-pod-container/configmap/

# Download the sample files into `configure-pod-container/
# configmap/` directory
wget https://kubernetes.io/examples/configmap/game.properties -O
configure-pod-container/configmap/game.properties
wget https://kubernetes.io/examples/configmap/ui.properties -O
configure-pod-container/configmap/ui.properties

# Create the configmap
kubectl create configmap game-config --from-file=configure-pod-
container/configmap/
```

The above command packages each file, in this case, `game.properties` and `ui.properties` in the `configure-pod-container/configmap/` directory into the `game-config ConfigMap`. You can display details of the `ConfigMap` using the following command:

```
kubectl describe configmaps game-config
```

The output is similar to this:

```
Name:          game-config
Namespace:     default
Labels:        <none>
Annotations:   <none>

Data
=====
game.properties:
-----
enemies=aliens
lives=3
enemies.cheat=true
enemies.cheat.level=noGoodRotten
secret.code.passphrase=UUDDLRLRBABAS
secret.code.allowed=true
secret.code.lives=30
ui.properties:
-----
color.good=purple
color.bad=yellow
allow.textmode=true
how.nice.to.look=fairlyNice
```

The `game.properties` and `ui.properties` files in the `configure-pod-container/configmap/` directory are represented in the data section of the `ConfigMap`.

```
kubectl get configmaps game-config -o yaml
```

The output is similar to this:

```
apiVersion: v1
kind: ConfigMap
metadata:
  creationTimestamp: 2016-02-18T18:52:05Z
  name: game-config
  namespace: default
  resourceVersion: "516"
```

```
uid: b4952dc3-d670-11e5-8cd0-68f728db1985
data:
  game.properties: |
    enemies=aliens
    lives=3
    enemies.cheat=true
    enemies.cheat.level=noGoodRotten
    secret.code.passphrase=UUDDLRLRBABAS
    secret.code.allowed=true
    secret.code.lives=30
  ui.properties: |
    color.good=purple
    color.bad=yellow
    allow.textmode=true
    how.nice.to.look=fairlyNice
```

## Create ConfigMaps from files

You can use `kubectl create configmap` to create a `ConfigMap` from an individual file, or from multiple files.

For example,

```
kubectl create configmap game-config-2 --from-file=configure-pod-
container/configmap/game.properties
```

would produce the following `ConfigMap`:

```
kubectl describe configmaps game-config-2
```

where the output is similar to this:

```
Name:          game-config-2
Namespace:     default
Labels:        <none>
Annotations:   <none>

Data
=====
game.properties:
-----
enemies=aliens
lives=3
enemies.cheat=true
enemies.cheat.level=noGoodRotten
```

```
secret.code.passphrase=UUDDLRLRBABAS  
secret.code.allowed=true  
secret.code.lives=30
```

You can pass in the `--from-file` argument multiple times to create a ConfigMap from multiple data sources.

```
kubectl create configmap game-config-2 --from-file=configure-pod-  
container/configmap/game.properties --from-file=configure-pod-  
container/configmap/ui.properties
```

You can display details of the `game-config-2` ConfigMap using the following command:

```
kubectl describe configmaps game-config-2
```

The output is similar to this:

```
Name:          game-config-2  
Namespace:    default  
Labels:        <none>  
Annotations:   <none>  
  
Data  
=====  
game.properties:  
----  
enemies=aliens  
lives=3  
enemies.cheat=true  
enemies.cheat.level=noGoodRotten  
secret.code.passphrase=UUDDLRLRBABAS  
secret.code.allowed=true  
secret.code.lives=30  
ui.properties:  
----  
color.good=purple  
color.bad=yellow  
allow.textmode=true  
how.nice.to.look=fairlyNice
```

Use the option `--from-env-file` to create a ConfigMap from an env-file, for example:

```
# Env-files contain a list of environment variables.  
# These syntax rules apply:
```

```

# Each line in an env file has to be in VAR=VAL format.
# Lines beginning with # (i.e. comments) are ignored.
# Blank lines are ignored.
# There is no special handling of quotation marks (i.e. they
will be part of the ConfigMap value).

# Download the sample files into `configure-pod-container/
configmap/` directory
wget https://kubernetes.io/examples/configmap/game-env-
file.properties -O configure-pod-container/configmap/game-env-
file.properties

# The env-file `game-env-file.properties` looks like below
cat configure-pod-container/configmap/game-env-file.properties
enemies=aliens
lives=3
allowed="true"

# This comment and the empty line above it are ignored

```

```

kubectl create configmap game-config-env-file \
--from-env-file=configure-pod-container/configmap/game-
env-file.properties

```

would produce the following ConfigMap:

```

kubectl get configmap game-config-env-file -o yaml

```

where the output is similar to this:

```

apiVersion: v1
kind: ConfigMap
metadata:
  creationTimestamp: 2017-12-27T18:36:28Z
  name: game-config-env-file
  namespace: default
  resourceVersion: "809965"
  uid: d9d1ca5b-eb34-11e7-887b-42010a8002b8
data:
  allowed: '"true"'
  enemies: aliens
  lives: "3"

```

**Caution:** When passing `--from-env-file` multiple times to create a ConfigMap from multiple data sources, only the last env-file is used.

The behavior of passing `--from-env-file` multiple times is demonstrated by:

```
# Download the sample files into `configure-pod-container/
configmap/` directory
wget https://kubernetes.io/examples/configmap/ui-env-
file.properties -O configure-pod-container/configmap/ui-env-
file.properties

# Create the configmap
kubectl create configmap config-multi-env-files \
    --from-env-file=configure-pod-container/configmap/game-
env-file.properties \
    --from-env-file=configure-pod-container/configmap/ui-env-
file.properties
```

would produce the following ConfigMap:

```
kubectl get configmap config-multi-env-files -o yaml
```

where the output is similar to this:

```
apiVersion: v1
kind: ConfigMap
metadata:
  creationTimestamp: 2017-12-27T18:38:34Z
  name: config-multi-env-files
  namespace: default
  resourceVersion: "810136"
  uid: 252c4572-eb35-11e7-887b-42010a8002b8
data:
  color: purple
  how: fairlyNice
  textmode: "true"
```

## Define the key to use when creating a ConfigMap from a file

You can define a key other than the file name to use in the data section of your ConfigMap when using the `--from-file` argument:

```
kubectl create configmap game-config-3 --from-file=<my-key-
name>=<path-to-file>
```

where `<my-key-name>` is the key you want to use in the ConfigMap and `<path-to-file>` is the location of the data source file you want the key to represent.

For example:

```
kubectl create configmap game-config-3 --from-file=game-special-key=configure-pod-container/configmap/game.properties
```

would produce the following ConfigMap:

```
kubectl get configmaps game-config-3 -o yaml
```

where the output is similar to this:

```
apiVersion: v1
kind: ConfigMap
metadata:
  creationTimestamp: 2016-02-18T18:54:22Z
  name: game-config-3
  namespace: default
  resourceVersion: "530"
  uid: 05f8da22-d671-11e5-8cd0-68f728db1985
data:
  game-special-key: |
    enemies=aliens
    lives=3
    enemies.cheat=true
    enemies.cheat.level=noGoodRotten
    secret.code.passphrase=UUDDLRLRBABAS
    secret.code.allowed=true
    secret.code.lives=30
```

## Create ConfigMaps from literal values

You can use `kubectl create configmap` with the `--from-literal` argument to define a literal value from the command line:

```
kubectl create configmap special-config --from-literal=special.how=very --from-literal=special.type=charm
```

You can pass in multiple key-value pairs. Each pair provided on the command line is represented as a separate entry in the `data` section of the ConfigMap.

```
kubectl get configmaps special-config -o yaml
```

The output is similar to this:

```
apiVersion: v1
kind: ConfigMap
metadata:
  creationTimestamp: 2016-02-18T19:14:38Z
  name: special-config
  namespace: default
  resourceVersion: "651"
  uid: dadce046-d673-11e5-8cd0-68f728db1985
data:
  special.how: very
  special.type: charm
```

## Create a ConfigMap from generator

*kubectl supports kustomization.yaml since 1.14. You can also create a ConfigMap from generators and then apply it to create the object on the Apiserver. The generators should be specified in a kustomization.yaml inside a directory.*

## Generate ConfigMaps from files

*For example, to generate a ConfigMap from files configure-pod-container/configmap/game.properties*

```
# Create a kustomization.yaml file with ConfigMapGenerator
cat <<EOF >./kustomization.yaml
configMapGenerator:
- name: game-config-4
  files:
  - configure-pod-container/configmap/game.properties
EOF
```

*Apply the kustomization directory to create the ConfigMap object.*

```
kubectl apply -k .
configmap/game-config-4-m9dm2f92bt created
```

*You can check that the ConfigMap was created like this:*

```

kubectl get configmap
NAME                DATA   AGE
game-config-4-m9dm2f92bt   1    37s

kubectl describe configmaps/game-config-4-m9dm2f92bt
Name:            game-config-4-m9dm2f92bt
Namespace:       default
Labels:          <none>
Annotations:    kubectl.kubernetes.io/last-applied-configuration:
                  {"apiVersion":"v1","data":{"game.properties":"enemies=aliens\nlives=3\nenemies.cheat=true\nenemies.cheat.level=noGoodRotten\nsecret.code.p...
Data
=====
game.properties:
-----
enemies=aliens
lives=3
enemies.cheat=true
enemies.cheat.level=noGoodRotten
secret.code.passphrase=UUDDLRLRBABAS
secret.code.allowed=true
secret.code.lives=30
Events:  <none>

```

*Note that the generated ConfigMap name has a suffix appended by hashing the contents. This ensures that a new ConfigMap is generated each time the content is modified.*

### **Define the key to use when generating a ConfigMap from a file**

*You can define a key other than the file name to use in the ConfigMap generator. For example, to generate a ConfigMap from files configure-pod-container/configmap/game.properties with the key game-special-key*

```

# Create a kustomization.yaml file with ConfigMapGenerator
cat <<EOF >./kustomization.yaml
configMapGenerator:
- name: game-config-5
  files:
  - game-special-key=configure-pod-container/configmap/
game.properties
EOF

```

*Apply the kustomization directory to create the ConfigMap object.*

```
kubectl apply -k .
configmap/game-config-5-m67dt67794 created
```

## Generate ConfigMaps from Literals

To generate a ConfigMap from literals `special.type=charm` and `special.how=very`, you can specify the ConfigMap generator in `kustomization.yaml` as

```
# Create a kustomization.yaml file with ConfigMapGenerator
cat <<EOF >./kustomization.yaml
configMapGenerator:
- name: special-config-2
  literals:
  - special.how=very
  - special.type=charm
EOF
```

Apply the kustomization directory to create the ConfigMap object.

```
kubectl apply -k .
configmap/special-config-2-c92b5mmcf2 created
```

## Define container environment variables using ConfigMap data

### Define a container environment variable with data from a single ConfigMap

1. Define an environment variable as a key-value pair in a ConfigMap:

```
kubectl create configmap special-config --from-literal=special.how=very
```

2. Assign the `special.how` value defined in the ConfigMap to the `SPECIAL_LEVEL_KEY` environment variable in the Pod specification.

[pods/pod-single-configmap-env-variable.yaml](#)

```
apiVersion: v1
kind: Pod
metadata:
  name: dapi-test-pod
```

```

spec:
  containers:
    - name: test-container
      image: k8s.gcr.io/busybox
      command: [ "/bin/sh", "-c", "env" ]
      env:
        # Define the environment variable
        - name: SPECIAL_LEVEL_KEY
          valueFrom:
            configMapKeyRef:
              # The ConfigMap containing the value you want to
              assign to SPECIAL_LEVEL_KEY
              name: special-config
              # Specify the key associated with the value
              key: special.how
      restartPolicy: Never

```

Create the Pod:

```
kubectl create -f https://kubernetes.io/examples/pods/pod-single-
configmap-env-variable.yaml
```

Now, the Pod's output includes environment variable `SPECIAL_LEVEL_KEY=very`.

## **Define container environment variables with data from multiple ConfigMaps**

- As with the previous example, create the ConfigMaps first.

[configmap/configmaps.yaml](#)



```

apiVersion: v1
kind: ConfigMap
metadata:
  name: special-config
  namespace: default
data:
  special.how: very
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: env-config
  namespace: default

```

```
data:  
  log_level: INFO
```

Create the ConfigMap:

```
kubectl create -f https://kubernetes.io/examples/configmap/  
configmaps.yaml
```

- Define the environment variables in the Pod specification.

[pods/pod-multiple-configmap-env-variable.yaml](#)  


```
apiVersion: v1  
kind: Pod  
metadata:  
  name: dapi-test-pod  
spec:  
  containers:  
    - name: test-container  
      image: k8s.gcr.io/busybox  
      command: [ "/bin/sh", "-c", "env" ]  
      env:  
        - name: SPECIAL_LEVEL_KEY  
          valueFrom:  
            configMapKeyRef:  
              name: special-config  
              key: special.how  
        - name: LOG_LEVEL  
          valueFrom:  
            configMapKeyRef:  
              name: env-config  
              key: log_level  
  restartPolicy: Never
```

Create the Pod:

```
kubectl create -f https://kubernetes.io/examples/pods/pod-  
multiple-configmap-env-variable.yaml
```

Now, the Pod's output includes environment variables `SPECIAL_LEVEL_KEY=very` and `LOG_LEVEL=INFO`.

## Configure all key-value pairs in a ConfigMap as container environment variables

**Note:** This functionality is available in Kubernetes v1.6 and later.

- Create a ConfigMap containing multiple key-value pairs.

[configmap/configmap-mutlikeys.yaml](#)  


```
apiVersion: v1
kind: ConfigMap
metadata:
  name: special-config
  namespace: default
data:
  SPECIAL_LEVEL: very
  SPECIAL_TYPE: charm
```

Create the ConfigMap:

```
kubectl create -f https://kubernetes.io/examples/configmap/
configmap-mutlikeys.yaml
```

- Use `envFrom` to define all of the ConfigMap's data as container environment variables. The key from the ConfigMap becomes the environment variable name in the Pod.

[pods/pod-configmap-envFrom.yaml](#)  


```
apiVersion: v1
kind: Pod
metadata:
  name: dapi-test-pod
spec:
  containers:
    - name: test-container
      image: k8s.gcr.io/busybox
      command: [ "/bin/sh", "-c", "env" ]
      envFrom:
        - configMapRef:
            name: special-config
  restartPolicy: Never
```

Create the Pod:

```
kubectl create -f https://kubernetes.io/examples/pods/pod-
configmap-envFrom.yaml
```

Now, the Pod's output includes environment variables `SPECIAL_LEVEL=very` and `SPECIAL_TYPE=charm`.

## Use ConfigMap-defined environment variables in Pod commands

You can use ConfigMap-defined environment variables in the `command` and `args` of a container using the `$(VAR_NAME)` Kubernetes substitution syntax.

For example, the following Pod specification

[pods/pod-configmap-env-var-valueFrom.yaml](#)  


```
apiVersion: v1
kind: Pod
metadata:
  name: dapi-test-pod
spec:
  containers:
    - name: test-container
      image: k8s.gcr.io/busybox
      command: [ "/bin/echo", "$(SPECIAL_LEVEL_KEY) $(SPECIAL_TYPE_KEY)" ]
      env:
        - name: SPECIAL_LEVEL_KEY
          valueFrom:
            configMapKeyRef:
              name: special-config
              key: SPECIAL_LEVEL
        - name: SPECIAL_TYPE_KEY
          valueFrom:
            configMapKeyRef:
              name: special-config
              key: SPECIAL_TYPE
  restartPolicy: Never
```

created by running

```
kubectl create -f https://kubernetes.io/examples/pods/pod-configmap-env-var-valueFrom.yaml
```

produces the following output in the `test-container` container:

`very charm`

# Add ConfigMap data to a Volume

As explained in [Create ConfigMaps from files](#), when you create a ConfigMap using `--from-file`, the filename becomes a key stored in the `data` section of the ConfigMap. The file contents become the key's value.

The examples in this section refer to a ConfigMap named `special-config`, shown below.

[configmap/configmap-multikeys.yaml](#)  


```
apiVersion: v1
kind: ConfigMap
metadata:
  name: special-config
  namespace: default
data:
  SPECIAL_LEVEL: very
  SPECIAL_TYPE: charm
```

Create the ConfigMap:

```
kubectl create -f https://kubernetes.io/examples/configmap/
configmap-multikeys.yaml
```

## Populate a Volume with data stored in a ConfigMap

Add the ConfigMap name under the `volumes` section of the Pod specification. This adds the ConfigMap data to the directory specified as `volumeMounts.mountPath` (in this case, `/etc/config`). The `command` section lists directory files with names that match the keys in ConfigMap.

[pods/pod-configmap-volume.yaml](#)  


```
apiVersion: v1
kind: Pod
metadata:
  name: dapi-test-pod
spec:
  containers:
    - name: test-container
      image: k8s.gcr.io/busybox
      command: [ "/bin/sh", "-c", "ls /etc/config/" ]
```

```

volumeMounts:
  - name: config-volume
    mountPath: /etc/config
volumes:
  - name: config-volume
configMap:
  # Provide the name of the ConfigMap containing the files
you want
  # to add to the container
  name: special-config
restartPolicy: Never

```

Create the Pod:

```
kubectl create -f https://kubernetes.io/examples/pods/pod-
configmap-volume.yaml
```

When the pod runs, the command `ls /etc/config/` produces the output below:

```
SPECIAL_LEVEL
SPECIAL_TYPE
```

**Caution:** If there are some files in the `/etc/config/` directory, they will be deleted.

**Note:** Text data is exposed as files using the UTF-8 character encoding. To use some other character encoding, use `binaryData`.

## Add ConfigMap data to a specific path in the Volume

Use the `path` field to specify the desired file path for specific ConfigMap items. In this case, the `SPECIAL_LEVEL` item will be mounted in the `config-volume` volume at `/etc/config/keys`.

[pods/pod-configmap-volume-specific-key.yaml](#)

```

apiVersion: v1
kind: Pod
metadata:
  name: dapi-test-pod
spec:
  containers:
    - name: test-container

```

```
image: k8s.gcr.io/busybox
command: [ "/bin/sh", "-c", "cat /etc/config/keys" ]
volumeMounts:
- name: config-volume
  mountPath: /etc/config
volumes:
- name: config-volume
  configMap:
    name: special-config
    items:
    - key: SPECIAL_LEVEL
      path: keys
restartPolicy: Never
```

Create the Pod:

```
kubectl create -f https://kubernetes.io/examples/pods/pod-
configmap-volume-specific-key.yaml
```

When the pod runs, the command `cat /etc/config/keys` produces the output below:

very

**Caution:** Like before, all previous files in the `/etc/config` directory will be deleted.

## **Project keys to specific paths and file permissions**

You can project keys to specific paths and specific permissions on a per-file basis. The [Secrets](#) user guide explains the syntax.

## **Mounted ConfigMaps are updated automatically**

When a ConfigMap already being consumed in a volume is updated, projected keys are eventually updated as well. Kubelet is checking whether the mounted ConfigMap is fresh on every periodic sync. However, it is using its local ttl-based cache for getting the current value of the ConfigMap. As a result, the total delay from the moment when the ConfigMap is updated to the moment when new keys are projected to the pod can be as long as kubelet sync period (1 minute by default) + ttl of ConfigMaps cache (1 minute by default) in kubelet. You can trigger an immediate refresh by updating one of the pod's annotations.

**Note:** A container using a ConfigMap as a [subPath](#) volume will not receive ConfigMap updates.

## Understanding ConfigMaps and Pods

The ConfigMap API resource stores configuration data as key-value pairs. The data can be consumed in pods or provide the configurations for system components such as controllers. ConfigMap is similar to [Secrets](#), but provides a means of working with strings that don't contain sensitive information. Users and system components alike can store configuration data in ConfigMap.

**Note:** ConfigMaps should reference properties files, not replace them. Think of the ConfigMap as representing something similar to the Linux /etc directory and its contents. For example, if you create a [Kubernetes Volume](#) from a ConfigMap, each data item in the ConfigMap is represented by an individual file in the volume.

The ConfigMap's data field contains the configuration data. As shown in the example below, this can be simple -- like individual properties defined using `--from-literal` -- or complex -- like configuration files or JSON blobs defined using `--from-file`.

```
apiVersion: v1
kind: ConfigMap
metadata:
  creationTimestamp: 2016-02-18T19:14:38Z
  name: example-config
  namespace: default
data:
  # example of a simple property defined using --from-literal
  example.property.1: hello
  example.property.2: world
  # example of a complex property defined using --from-file
  example.property.file: |-
    property.1=value-1
    property.2=value-2
    property.3=value-3
```

## Restrictions

- You must create a ConfigMap before referencing it in a Pod specification (unless you mark the ConfigMap as "optional"). If you reference a ConfigMap that doesn't exist, the Pod won't start. Likewise,

references to keys that don't exist in the ConfigMap will prevent the pod from starting.

- If you use `envFrom` to define environment variables from ConfigMaps, keys that are considered invalid will be skipped. The pod will be allowed to start, but the invalid names will be recorded in the event log (`InvalidVariableNames`). The log message lists each skipped key. For example:

```
kubectl get events
```

The output is similar to this:

LASTSEEN	FIRSTSEEN	COUNT	NAME	KIND	SUBOBJECT
TYPE			REASON		
SOURCE			MESSAGE		
0s	0s	1	dapi-test-pod Pod		
Warning			InvalidEnvironmentVariableNames {kubelet, 127.0.0.1} Keys [1badkey, 2alsobad] from the EnvFrom configMap default/myconfig were skipped since they are considered invalid environment variable names.		

- ConfigMaps reside in a specific [Namespace](#). A ConfigMap can only be referenced by pods residing in the same namespace.
- You can't use ConfigMaps for [static pods](#), because the Kubelet does not support this.

## What's next

- Follow a real world example of [Configuring Redis using a ConfigMap](#).

## 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](#).

Last modified November 17, 2020 at 9:54 PM PST: [Improve configmap usage as pod command and args \(1ec78b1f2\)](#)  
[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Create a ConfigMap](#)
  - [Create a ConfigMap Using kubectl create configmap](#)

- [Create a ConfigMap from generator](#)
- [Define container environment variables using ConfigMap data](#)
  - [Define a container environment variable with data from a single ConfigMap](#)
  - [Define container environment variables with data from multiple ConfigMaps](#)
- [Configure all key-value pairs in a ConfigMap as container environment variables](#)
- [Use ConfigMap-defined environment variables in Pod commands](#)
- [Add ConfigMap data to a Volume](#)
  - [Populate a Volume with data stored in a ConfigMap](#)
  - [Add ConfigMap data to a specific path in the Volume](#)
  - [Project keys to specific paths and file permissions](#)
  - [Mounted ConfigMaps are updated automatically](#)
- [Understanding ConfigMaps and Pods](#)
  - [Restrictions](#)
- [What's next](#)

# **Share Process Namespace between Containers in a Pod**

**FEATURE STATE:** Kubernetes v1.17 [stable]

*This page shows how to configure process namespace sharing for a pod. When process namespace sharing is enabled, processes in a container are visible to all other containers in that pod.*

*You can use this feature to configure cooperating containers, such as a log handler sidecar container, or to troubleshoot container images that don't include debugging utilities like a shell.*

## **Before you begin**

*You need to have a Kubernetes cluster, and the `kubectl` command-line tool must be configured to communicate with your cluster. If you do not already have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:*

- [Katacoda](#)
- [Play with Kubernetes](#)

*Your Kubernetes server must be at or later than version v1.10. To check the version, enter `kubectl version`.*

## **Configure a Pod**

*Process Namespace Sharing is enabled using the `shareProcessNamespace` field of `v1.PodSpec`. For example:*

[pods/share-process-namespace.yaml](#)  
[

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  shareProcessNamespace: true
  containers:
    - name: nginx
      image: nginx
    - name: shell
      image: busybox
  securityContext:
    capabilities:
      add:
        - SYS_PTRACE
  stdin: true
  tty: true
```

1. Create the pod `nginx` on your cluster:

```
kubectl apply -f https://k8s.io/examples/pods/share-process-
namespace.yaml
```

2. Attach to the `shell` container and run `ps`:

```
kubectl attach -it nginx -c shell
```

If you don't see a command prompt, try pressing enter.

```
/ # ps ax
 PID  USER      TIME  COMMAND
   1  root      0:00 /pause
   8  root      0:00 nginx: master process nginx -g daemon
off;
  14  101      0:00 nginx: worker process
  15  root      0:00 sh
  21  root      0:00 ps ax
```

You can signal processes in other containers. For example, send `SIGHUP` to `nginx` to restart the worker process. This requires the `SYS_PTRACE` capability.

```
/ # kill -HUP 8
/ # ps ax
PID  USER      TIME  COMMAND
```

```
1 root      0:00 /pause
8 root      0:00 nginx: master process nginx -g daemon off;
15 root     0:00 sh
22 101      0:00 nginx: worker process
23 root     0:00 ps ax
```

*It's even possible to access another container image using the `/proc/$pid/root` link.*

```
/ # head /proc/8/root/etc/nginx/nginx.conf

user nginx;
worker_processes 1;

error_log /var/log/nginx/error.log warn;
pid        /var/run/nginx.pid;

events {
    worker_connections 1024;
```

## **Understanding Process Namespace Sharing**

*Pods share many resources so it makes sense they would also share a process namespace. Some container images may expect to be isolated from other containers, though, so it's important to understand these differences:*

- 1. The container process no longer has PID 1.** Some container images refuse to start without PID 1 (for example, containers using `systemd`) or run commands like `kill -HUP 1` to signal the container process. In pods with a shared process namespace, `kill -HUP 1` will signal the pod sandbox. (`/pause` in the above example.)
- 2. Processes are visible to other containers in the pod.** This includes all information visible in `/proc`, such as passwords that were passed as arguments or environment variables. These are protected only by regular Unix permissions.
- 3. Container filesystems are visible to other containers in the pod through the `/proc/$pid/root` link.** This makes debugging easier, but it also means that filesystem secrets are protected only by filesystem permissions.

## **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](#).

Last modified May 30, 2020 at 3:10 PM PST: [add en pages \(ecc27bbbe\)](#)  
[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Configure a Pod](#)
- [Understanding Process Namespace Sharing](#)

## Create static Pods

Static Pods are managed directly by the kubelet daemon on a specific node, without the [API server](#) observing them. Unlike Pods that are managed by the control plane (for example, a [Deployment](#)); instead, the kubelet watches each static Pod (and restarts it if it fails).

Static Pods are always bound to one [Kubelet](#) on a specific node.

The kubelet automatically tries to create a [mirror Pod](#) on the Kubernetes API server for each static Pod. This means that the Pods running on a node are visible on the API server, but cannot be controlled from there.

**Note:** If you are running clustered Kubernetes and are using static Pods to run a Pod on every node, you should probably be using a [DaemonSet](#) instead.

## Before you begin

You need to have a Kubernetes cluster, and the kubectl command-line tool must be configured to communicate with your cluster. If you do not already have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:

- [Katacoda](#)
- [Play with Kubernetes](#)

To check the version, enter `kubectl version`.

This page assumes you're using [Docker](#) to run Pods, and that your nodes are running the Fedora operating system. Instructions for other distributions or Kubernetes installations may vary.

## Create a static pod

You can configure a static Pod with either a [file system hosted configuration file](#) or a [web hosted configuration file](#).

## Filesystem-hosted static Pod manifest

Manifests are standard Pod definitions in JSON or YAML format in a specific directory. Use the `staticPodPath: <the directory>` field in the [kubelet configuration file](#), which periodically scans the directory and creates/deletes static Pods as YAML/JSON files appear/disappear there. Note that the kubelet will ignore files starting with dots when scanning the specified directory.

For example, this is how to start a simple web server as a static Pod:

1. Choose a node where you want to run the static Pod. In this example, it's `my-node1`.

```
ssh my-node1
```

2. Choose a directory, say `/etc/kubelet.d` and place a web server Pod definition there, for example `/etc/kubelet.d/static-web.yaml`:

```
# Run this command on the node where kubelet is running
mkdir /etc/kubelet.d/
cat <<EOF >/etc/kubelet.d/static-web.yaml
apiVersion: v1
kind: Pod
metadata:
  name: static-web
  labels:
    role: myrole
spec:
  containers:
    - name: web
      image: nginx
      ports:
        - name: web
          containerPort: 80
          protocol: TCP
EOF
```

3. Configure your kubelet on the node to use this directory by running it with `--pod-manifest-path=/etc/kubelet.d/` argument. On Fedora edit `/etc/kubernetes/kubelet` to include this line:

```
KUBELET_ARGS="--cluster-dns=10.254.0.10 --cluster-
domain=kube.local --pod-manifest-path=/etc/kubelet.d/"
```

or add the `staticPodPath: <the directory>` field in the [kubelet configuration file](#).

4. Restart the kubelet. On Fedora, you would run:

```
# Run this command on the node where the kubelet is running
systemctl restart kubelet
```

## Web-hosted static pod manifest

Kubelet periodically downloads a file specified by `--manifest-url=<URL>` argument and interprets it as a JSON/YAML file that contains Pod definitions. Similar to how [filesystem-hosted manifests](#) work, the kubelet refetches the manifest on a schedule. If there are changes to the list of static Pods, the kubelet applies them.

To use this approach:

1. Create a YAML file and store it on a web server so that you can pass the URL of that file to the kubelet.

```
apiVersion: v1
kind: Pod
metadata:
  name: static-web
  labels:
    role: myrole
spec:
  containers:
    - name: web
      image: nginx
      ports:
        - name: web
          containerPort: 80
          protocol: TCP
```

2. Configure the kubelet on your selected node to use this web manifest by running it with `--manifest-url=<manifest-url>`. On Fedora, edit `/etc/kubernetes/kubelet` to include this line:

```
KUBELET_ARGS="--cluster-dns=10.254.0.10 --cluster-
domain=kube.local --manifest-url=<manifest-url>"
```

3. Restart the kubelet. On Fedora, you would run:

```
# Run this command on the node where the kubelet is running
systemctl restart kubelet
```

## **Observe static pod behavior**

*When the kubelet starts, it automatically starts all defined static Pods. As you have defined a static Pod and restarted the kubelet, the new static Pod should already be running.*

*You can view running containers (including static Pods) by running (on the node):*

```
# Run this command on the node where the kubelet is running
docker ps
```

*The output might be something like:*

COUNTAINER ID	IMAGE	COMMAND	CREATED
STATUS	PORTS	NAMES	
f6d05272b57e	nginx:latest	"nginx"	8 minutes ago
minutes		k8s_web.6f802af4_static-web-fk-	
node1_default_67e24ed9466ba55986d120c867395f3c_378e5f3c			

*You can see the mirror Pod on the API server:*

```
kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
static-web-my-node1	1/1	Running	0	2m

**Note:** Make sure the kubelet has permission to create the mirror Pod in the API server. If not, the creation request is rejected by the API server. See [PodSecurityPolicy](#).

*Labels from the static Pod are propagated into the mirror Pod. You can use those labels as normal via [selectors](#), etc.*

*If you try to use kubectl to delete the mirror Pod from the API server, the kubelet doesn't remove the static Pod:*

```
kubectl delete pod static-web-my-node1
```

```
pod "static-web-my-node1" deleted
```

You can see that the Pod is still running:

```
kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
static-web-my-node1	1/1	Running	0	12s

Back on your node where the kubelet is running, you can try to stop the Docker container manually. You'll see that, after a time, the kubelet will notice and will restart the Pod automatically:

```
# Run these commands on the node where the kubelet is running
docker stop f6d05272b57e # replace with the ID of your container
sleep 20
docker ps
```

CONTAINER ID	IMAGE	COMMAND
CREATED	...	
5b920cba8b1	nginx:latest	"nginx -g 'daemon off;' 2 seconds ago ..."

## **Dynamic addition and removal of static pods**

The running kubelet periodically scans the configured directory (`/etc/kubelet.d` in our example) for changes and adds/removes Pods as files appear/disappear in this directory.

```
# This assumes you are using filesystem-hosted static Pod
# configuration
# Run these commands on the node where the kubelet is running
#
mv /etc/kubelet.d/static-web.yaml /tmp
sleep 20
docker ps
# You see that no nginx container is running
mv /tmp/static-web.yaml /etc/kubelet.d/
sleep 20
docker ps
```

CONTAINER ID	IMAGE	COMMAND
CREATED	...	
e7a62e3427f1	nginx:latest	"nginx -g 'daemon off;' 27 seconds ago ..."

# 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](#).

Last modified July 27, 2020 at 4:18 AM PST: [Revise Pod concept \(#22603\)](#) ([49eee8fd3](#))

[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Create a static pod](#)
  - [Filesystem-hosted static Pod manifest](#)
  - [Web-hosted static pod manifest](#)
- [Observe static pod behavior](#)
- [Dynamic addition and removal of static pods](#)

# Translate a Docker Compose File to Kubernetes Resources

What's Kompose? It's a conversion tool for all things compose (namely Docker Compose) to container orchestrators (Kubernetes or OpenShift).

More information can be found on the Kompose website at <http://kompose.io>.

## Before you begin

You need to have a Kubernetes cluster, and the kubectl command-line tool must be configured to communicate with your cluster. If you do not already have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:

- [Katacoda](#)
- [Play with Kubernetes](#)

To check the version, enter `kubectl version`.

## Install Kompose

We have multiple ways to install Kompose. Our preferred method is downloading the binary from the latest GitHub release.

- [GitHub download](#)
- [Build from source](#)
- [CentOS package](#)
- [Fedora package](#)
- [Homebrew \(macOS\)](#)

*Kompose is released via GitHub on a three-week cycle, you can see all current releases on the [GitHub release page](#).*

```
# Linux
curl -L https://github.com/kubernetes/kompose/releases/download/v1.21.0/kompose-linux-amd64 -o kompose

# macOS
curl -L https://github.com/kubernetes/kompose/releases/download/v1.21.0/kompose-darwin-amd64 -o kompose

# Windows
curl -L https://github.com/kubernetes/kompose/releases/download/v1.21.0/kompose-windows-amd64.exe -o kompose.exe

chmod +x kompose
sudo mv ./kompose /usr/local/bin/kompose
```

*Alternatively, you can download the [tarball](#).*

*Installing using go get pulls from the master branch with the latest development changes.*

```
go get -u github.com/kubernetes/kompose
```

*Kompose is in [EPEL](#) CentOS repository. If you don't have [EPEL](#) repository already installed and enabled you can do it by running sudo yum install epel-release*

*If you have [EPEL](#) enabled in your system, you can install Kompose like any other package.*

```
sudo yum -y install kompose
```

*Kompose is in Fedora 24, 25 and 26 repositories. You can install it just like any other package.*

```
sudo dnf -y install kompose
```

*On macOS you can install latest release via [Homebrew](#):*

```
brew install kompose
```

# Use Kompose

In just a few steps, we'll take you from Docker Compose to Kubernetes. All you need is an existing `docker-compose.yml` file.

1. Go to the directory containing your `docker-compose.yml` file. If you don't have one, test using this one.

```
version: "2"

services:

  redis-master:
    image: k8s.gcr.io/redis:e2e
    ports:
      - "6379"

  redis-slave:
    image: gcr.io/google_samples/gb-redisslave:v3
    ports:
      - "6379"
    environment:
      - GET_HOSTS_FROM=dns

  frontend:
    image: gcr.io/google-samples/gb-frontend:v4
    ports:
      - "80:80"
    environment:
      - GET_HOSTS_FROM=dns
    labels:
      kompose.service.type: LoadBalancer
```

2. Run the `kompose up` command to deploy to Kubernetes directly, or skip to the next step instead to generate a file to use with `kubectl`.

```
$ kompose up
We are going to create Kubernetes Deployments, Services and
PersistentVolumeClaims for your Dockerized application.
If you need different kind of resources, use the 'kompose
convert' and 'kubectl apply -f' commands instead.
```

```
INFO Successfully created Service: redis
INFO Successfully created Service: web
INFO Successfully created Deployment: redis
INFO Successfully created Deployment: web
```

```
Your application has been deployed to Kubernetes. You can
run 'kubectl get deployment,svc,pods,pvc' for details.
```

3. To convert the `docker-compose.yml` file to files that you can use with `kubectl`, run `kompose convert` and then `kubectl apply -f <output file>`.

```
$ kompose convert
INFO Kubernetes file "frontend-service.yaml" created
INFO Kubernetes file "redis-master-service.yaml" created
INFO Kubernetes file "redis-slave-service.yaml" created
INFO Kubernetes file "frontend-deployment.yaml" created
INFO Kubernetes file "redis-master-deployment.yaml" created
INFO Kubernetes file "redis-slave-deployment.yaml" created

$ kubectl apply -f frontend-service.yaml,redis-master-
service.yaml,redis-slave-service.yaml,frontend-
deployment.yaml,redis-master-deployment.yaml,redis-slave-
deployment.yaml
service/frontend created
service/redis-master created
service/redis-slave created
deployment.apps/frontend created
deployment.apps/redis-master created
deployment.apps/redis-slave created
```

Your deployments are running in Kubernetes.

4. Access your application.

If you're already using `minikube` for your development process:

```
$ minikube service frontend
```

Otherwise, let's look up what IP your service is using!

```
$ kubectl describe svc frontend
Name:           frontend
Namespace:      default
Labels:         service=frontend
Selector:       service=frontend
Type:          LoadBalancer
IP:            10.0.0.183
LoadBalancer Ingress:  192.0.2.89
Port:          80      80/TCP
NodePort:      80      31144/TCP
Endpoints:     172.17.0.4:80
Session Affinity: None
No events.
```

If you're using a cloud provider, your IP will be listed next to `LoadBalancer Ingress`.

```
$ curl http://192.0.2.89
```

# User Guide

- *CLI*
  - [kompose convert](#)
  - [kompose up](#)
  - [kompose down](#)
- *Documentation*
  - [Build and Push Docker Images](#)
  - [Alternative Conversions](#)
  - [Labels](#)
  - [Restart](#)
  - [Docker Compose Versions](#)

*Kompose has support for two providers: OpenShift and Kubernetes. You can choose a targeted provider using global option `--provider`. If no provider is specified, Kubernetes is set by default.*

## **`kompose convert`**

*Kompose supports conversion of V1, V2, and V3 Docker Compose files into Kubernetes and OpenShift objects.*

## **Kubernetes**

```
$ kompose --file docker-voting.yml convert
WARN Unsupported key networks - ignoring
WARN Unsupported key build - ignoring
INFO Kubernetes file "worker-svc.yaml" created
INFO Kubernetes file "db-svc.yaml" created
INFO Kubernetes file "redis-svc.yaml" created
INFO Kubernetes file "result-svc.yaml" created
INFO Kubernetes file "vote-svc.yaml" created
INFO Kubernetes file "redis-deployment.yaml" created
INFO Kubernetes file "result-deployment.yaml" created
INFO Kubernetes file "vote-deployment.yaml" created
INFO Kubernetes file "worker-deployment.yaml" created
INFO Kubernetes file "db-deployment.yaml" created

$ ls
db-deployment.yaml docker-compose.yml docker-
gitlab.yml redis-deployment.yaml result-deployment.yaml vote-
deployment.yaml worker-deployment.yaml
db-svc.yaml docker-voting.yml redis-
svc.yaml result-svc.yaml vote-svc.yaml
worker-svc.yaml
```

You can also provide multiple docker-compose files at the same time:

```
$ kompose -f docker-compose.yml -f docker-guestbook.yml convert
INFO Kubernetes file "frontend-service.yaml" created
INFO Kubernetes file "mlbparks-service.yaml" created
INFO Kubernetes file "mongodb-service.yaml" created
INFO Kubernetes file "redis-master-service.yaml" created
INFO Kubernetes file "redis-slave-service.yaml" created
INFO Kubernetes file "frontend-deployment.yaml" created
INFO Kubernetes file "mlbparks-deployment.yaml" created
INFO Kubernetes file "mongodb-deployment.yaml" created
INFO Kubernetes file "mongodb-claim0-persistentvolumeclaim.yaml"
created
INFO Kubernetes file "redis-master-deployment.yaml" created
INFO Kubernetes file "redis-slave-deployment.yaml" created

$ ls
mlbparks-deployment.yaml  mongodb-
service.yaml               redis-slave-
service.jsonmlbparks-service.yaml
frontend-deployment.yaml  mongodb-claim0-
persistentvolumeclaim.yaml redis-master-service.yaml
frontend-service.yaml      mongodb-
deployment.yaml           redis-slave-deployment.yaml
redis-master-deployment.yaml
```

When multiple docker-compose files are provided the configuration is merged. Any configuration that is common will be over ridden by subsequent file.

## OpenShift

```
$ kompose --provider openshift --file docker-voting.yml convert
WARN [worker] Service cannot be created because of missing port.
INFO OpenShift file "vote-service.yaml" created
INFO OpenShift file "db-service.yaml" created
INFO OpenShift file "redis-service.yaml" created
INFO OpenShift file "result-service.yaml" created
INFO OpenShift file "vote-deploymentconfig.yaml" created
INFO OpenShift file "vote-imagestream.yaml" created
INFO OpenShift file "worker-deploymentconfig.yaml" created
INFO OpenShift file "worker-imagestream.yaml" created
INFO OpenShift file "db-deploymentconfig.yaml" created
INFO OpenShift file "db-imagestream.yaml" created
INFO OpenShift file "redis-deploymentconfig.yaml" created
INFO OpenShift file "redis-imagestream.yaml" created
INFO OpenShift file "result-deploymentconfig.yaml" created
INFO OpenShift file "result-imagestream.yaml" created
```

*It also supports creating buildconfig for build directive in a service. By default, it uses the remote repo for the current git branch as the source repo, and the current branch as the source branch for the build. You can specify a different source repo and branch using --build-repo and --build-branch options respectively.*

```
$ kompose --provider openshift --file buildconfig/docker-compose.yml convert
WARN [foo] Service cannot be created because of missing port.
INFO OpenShift Buildconfig using git@github.com:rtnpro/kompose.git::master as source.
INFO OpenShift file "foo-deploymentconfig.yaml" created
INFO OpenShift file "foo-imagestream.yaml" created
INFO OpenShift file "foo-buildconfig.yaml" created
```

**Note:** If you are manually pushing the OpenShift artifacts using `oc create -f`, you need to ensure that you push the imagestream artifact before the buildconfig artifact, to workaround this OpenShift issue: <https://github.com/openshift/origin/issues/4518> .

## **kompose up**

Kompose supports a straightforward way to deploy your "composed" application to Kubernetes or OpenShift via `kompose up`.

### **Kubernetes**

```
$ kompose --file ./examples/docker-guestbook.yml up
We are going to create Kubernetes deployments and services for
your Dockerized application.
If you need different kind of resources, use the 'kompose
convert' and 'kubectl apply -f' commands instead.
```

```
INFO Successfully created service: redis-master
INFO Successfully created service: redis-slave
INFO Successfully created service: frontend
INFO Successfully created deployment: redis-master
INFO Successfully created deployment: redis-slave
INFO Successfully created deployment: frontend
```

Your application has been deployed to Kubernetes. You can run 'kubectl get deployment,svc,pods' for details.

NAME	DESIRED

CURRENT	UP-TO-DATE	AVAILABLE	AGE		
deployment.extensions/frontend	1	1	4m	1	1
deployment.extensions/redis-master	1	1	4m	1	1
deployment.extensions/redis-slave	1	1	4m	1	1
NAME	TYPE	CLUSTER-IP			
EXTERNAL-IP	PORT(S)	AGE			
service/frontend		ClusterIP	10.0.174.12		
<none>	80/TCP	4m			
service/kubernetes		ClusterIP	10.0.0.1		
<none>	443/TCP	13d			
service/redis-master		ClusterIP	10.0.202.43		
<none>	6379/TCP	4m			
service/redis-slave		ClusterIP	10.0.1.85		
<none>	6379/TCP	4m			
NAME	READY	STATUS			
RESTARTS	AGE				
pod/frontend-2768218532-cs5t5	1/1	Running	0		
4m					
pod/redis-master-1432129712-63jn8	1/1	Running	0		
4m					
pod/redis-slave-2504961300-nve7b	1/1	Running	0		
4m					

### Note:

- You must have a running Kubernetes cluster with a pre-configured `kubectl` context.
- Only deployments and services are generated and deployed to Kubernetes. If you need different kind of resources, use the `kompose convert` and `kubectl apply -f` commands instead.

## OpenShift

```
$ kompose --file ./examples/docker-guestbook.yml --provider openshift up
We are going to create OpenShift DeploymentConfigs and Services f or your Dockerized application.
If you need different kind of resources, use the 'kompose convert' and 'oc create -f' commands instead.

INFO Successfully created service: redis-slave
INFO Successfully created service: frontend
INFO Successfully created service: redis-master
```

```

INFO Successfully created deployment: redis-slave
INFO Successfully created ImageStream: redis-slave
INFO Successfully created deployment: frontend
INFO Successfully created ImageStream: frontend
INFO Successfully created deployment: redis-master
INFO Successfully created ImageStream: redis-master

```

*Your application has been deployed to OpenShift. You can run '`oc get dc,svc,is`' for details.*

```

$ oc get dc,svc,is
NAME                      REVISION
DESIRED      CURRENT      TRIGGERED BY
dc/frontend    0           config,image(frontend:v4)          1
dc/redis-master 0           config,image(redis-master:e2e)       1
dc/redis-slave   0           config,image(redis-slave:v1)        1
NAME                      CLUSTER-IP
EXTERNAL-IP    PORT(S)     AGE
svc/frontend   172.30.46.64
<none>         80/TCP      8s
svc/redis-master 172.30.144.56
<none>         6379/TCP    8s
svc/redis-slave 172.30.75.245
<none>         6379/TCP    8s
NAME                      DOCKER REPO
TAGS              UPDATED
is/frontend      172.30.12.200:5000/fff/
frontend
is/redis-master  172.30.12.200:5000/fff/redis-
master
is/redis-slave   172.30.12.200:5000/fff/redis-slave      v1

```

### Note:

- You must have a running OpenShift cluster with a pre-configured `oc` context (`oc login`)

## **kompose down**

Once you have deployed "composed" application to Kubernetes, `$ kompose down` will help you to take the application out by deleting its deployments and services. If you need to remove other resources, use the 'kubectl' command.

```
$ kompose --file docker-guestbook.yml down
INFO Successfully deleted service: redis-master
INFO Successfully deleted deployment: redis-master
INFO Successfully deleted service: redis-slave
INFO Successfully deleted deployment: redis-slave
INFO Successfully deleted service: frontend
INFO Successfully deleted deployment: frontend
```

**Note:**

- You must have a running Kubernetes cluster with a pre-configured kubectl context.

## Build and Push Docker Images

Kompose supports both building and pushing Docker images. When using the `build` key within your Docker Compose file, your image will:

- Automatically be built with Docker using the `image` key specified within your file
- Be pushed to the correct Docker repository using local credentials (located at `.docker/config`)

Using an [example Docker Compose file](#):

```
version: "2"

services:
  foo:
    build: "./build"
    image: docker.io/foo/bar
```

Using `kompose up` with a `build` key:

```
$ kompose up
INFO Build key detected. Attempting to build and push image
'docker.io/foo/bar'
INFO Building image 'docker.io/foo/bar' from directory 'build'
INFO Image 'docker.io/foo/bar' from directory 'build' built
successfully
INFO Pushing image 'foo/bar:latest' to registry 'docker.io'
INFO Attempting authentication credentials 'https://
index.docker.io/v1/'
INFO Successfully pushed image 'foo/bar:latest' to registry
```

```
'docker.io'  
INFO We are going to create Kubernetes Deployments, Services and PersistentVolumeClaims for your Dockerized application. If you need different kind of resources, use the 'kompose convert' and 'kubectl apply -f' commands instead.
```

```
INFO Deploying application in "default" namespace  
INFO Successfully created Service: foo  
INFO Successfully created Deployment: foo
```

Your application has been deployed to Kubernetes. You can run 'kubectl get deployment,svc,pods,pvc' for details.

In order to disable the functionality, or choose to use BuildConfig generation (with OpenShift) --build (local/build-config/none) can be passed.

```
# Disable building/pushing Docker images  
$ kompose up --build none  
  
# Generate Build Config artifacts for OpenShift  
$ kompose up --provider openshift --build build-config
```

## Alternative Conversions

The default kompose transformation will generate Kubernetes [Deployments](#) and [Services](#), in yaml format. You have alternative option to generate json with -j. Also, you can alternatively generate [Replication Controllers](#) objects, [Daemon Sets](#), or [Helm](#) charts.

```
$ kompose convert -j  
INFO Kubernetes file "redis-svc.json" created  
INFO Kubernetes file "web-svc.json" created  
INFO Kubernetes file "redis-deployment.json" created  
INFO Kubernetes file "web-deployment.json" created
```

The \*-deployment.json files contain the Deployment objects.

```
$ kompose convert --replication-controller  
INFO Kubernetes file "redis-svc.yaml" created  
INFO Kubernetes file "web-svc.yaml" created  
INFO Kubernetes file "redis-replicationcontroller.yaml" created  
INFO Kubernetes file "web-replicationcontroller.yaml" created
```

The `*-replicationcontroller.yaml` files contain the Replication Controller objects. If you want to specify replicas (default is 1), use `--replicas` flag: \$ kompose convert --replication-controller --replicas 3

```
$ kompose convert --daemon-set
INFO Kubernetes file "redis-svc.yaml" created
INFO Kubernetes file "web-svc.yaml" created
INFO Kubernetes file "redis-daemonset.yaml" created
INFO Kubernetes file "web-daemonset.yaml" created
```

The `*-daemonset.yaml` files contain the Daemon Set objects

If you want to generate a Chart to be used with [Helm](#) simply do:

```
$ kompose convert -c
INFO Kubernetes file "web-svc.yaml" created
INFO Kubernetes file "redis-svc.yaml" created
INFO Kubernetes file "web-deployment.yaml" created
INFO Kubernetes file "redis-deployment.yaml" created
chart created in "./docker-compose/"

$ tree docker-compose/
docker-compose
└── Chart.yaml
    ├── README.md
    └── templates
        ├── redis-deployment.yaml
        ├── redis-svc.yaml
        ├── web-deployment.yaml
        └── web-svc.yaml
```

The chart structure is aimed at providing a skeleton for building your Helm charts.

## Labels

`kompose` supports Kompose-specific labels within the `docker-compose.yml` file in order to explicitly define a service's behavior upon conversion.

- `kompose.service.type` defines the type of service to be created.

For example:

```

version: "2"
services:
  nginx:
    image: nginx
    dockerfile: foobar
    build: ./foobar
    cap_add:
      - ALL
    container_name: foobar
    labels:
      kompose.service.type: nodeport

```

- *kompose.service.expose* defines if the service needs to be made accessible from outside the cluster or not. If the value is set to "true", the provider sets the endpoint automatically, and for any other value, the value is set as the hostname. If multiple ports are defined in a service, the first one is chosen to be the exposed.
  - For the Kubernetes provider, an ingress resource is created and it is assumed that an ingress controller has already been configured.
  - For the OpenShift provider, a route is created.

For example:

```

version: "2"
services:
  web:
    image: tuna/docker-counter23
    ports:
      - "5000:5000"
    links:
      - redis
    labels:
      kompose.service.expose: "counter.example.com"
  redis:
    image: redis:3.0
    ports:
      - "6379"

```

The currently supported options are:

Key	Value
kompose.service.type	nodeport / clusterip / loadbalancer
kompose.service.expose	true / hostname

**Note:** The *kompose.service.type* label should be defined with *ports* only, otherwise *kompose* will fail.

## **Restart**

If you want to create normal pods without controllers you can use `restart` construct of docker-compose to define that. Follow table below to see what happens on the `restart` value.

<b><code>docker-compose restart</code></b>	<b><code>object created</code></b>	<b><code>Pod restartPolicy</code></b>
<code>""</code>	<code>controller object</code>	<code>Always</code>
<code>always</code>	<code>controller object</code>	<code>Always</code>
<code>on-failure</code>	<code>Pod</code>	<code>OnFailure</code>
<code>no</code>	<code>Pod</code>	<code>Never</code>

**Note:** The controller object could be deployment or replication controller, etc.

For example, the `pival` service will become pod down here. This container calculated value of `pi`.

```
version: '2'

services:
  pival:
    image: perl
    command: ["perl", "-Mbignum=bpi", "-wle", "print bpi(2000)"]
    restart: "on-failure"
```

## **Warning about Deployment Config's**

If the Docker Compose file has a volume specified for a service, the Deployment (Kubernetes) or DeploymentConfig (OpenShift) strategy is changed to "Recreate" instead of "RollingUpdate" (default). This is done to avoid multiple instances of a service from accessing a volume at the same time.

If the Docker Compose file has service name with `_` in it (eg.`web_service`), then it will be replaced by `-` and the service name will be renamed accordingly (eg.`web-service`). Kompose does this because "Kubernetes" doesn't allow `_` in object name.

Please note that changing service name might break some `docker-compose` files.

# Docker Compose Versions

Kompose supports Docker Compose versions: 1, 2 and 3. We have limited support on versions 2.1 and 3.2 due to their experimental nature.

A full list on compatibility between all three versions is listed in our [conversion document](#) including a list of all incompatible Docker Compose keys.

## 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](#).

Last modified May 30, 2020 at 3:10 PM PST: [add en pages \(ecc27bbbe\)](#)  
[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Install Kompose](#)
- [Use Kompose](#)
- [User Guide](#)
- [kompose convert](#)
  - [Kubernetes](#)
  - [OpenShift](#)
- [kompose up](#)
  - [Kubernetes](#)
  - [OpenShift](#)
- [kompose down](#)
- [Build and Push Docker Images](#)
- [Alternative Conversions](#)
- [Labels](#)
- [Restart](#)
  - [Warning about Deployment Config's](#)
- [Docker Compose Versions](#)

# Manage Kubernetes Objects

Declarative and imperative paradigms for interacting with the Kubernetes API.

---

[Declarative Management of Kubernetes Objects Using Configuration Files](#)

## [Declarative Management of Kubernetes Objects Using Kustomize](#)

## [Managing Kubernetes Objects Using Imperative Commands](#)

## [Imperative Management of Kubernetes Objects Using Configuration Files](#)

### [Update API Objects in Place Using kubectl patch](#)

Use `kubectl patch` to update Kubernetes API objects in place. Do a strategic merge patch or a JSON merge patch.

# **Declarative Management of Kubernetes Objects Using Configuration Files**

Kubernetes objects can be created, updated, and deleted by storing multiple object configuration files in a directory and using `kubectl apply` to recursively create and update those objects as needed. This method retains writes made to live objects without merging the changes back into the object configuration files. `kubectl diff` also gives you a preview of what changes `apply` will make.

## **Before you begin**

Install [`kubectl`](#).

You need to have a Kubernetes cluster, and the `kubectl` command-line tool must be configured to communicate with your cluster. If you do not already have a cluster, you can create one by using [`minikube`](#) or you can use one of these Kubernetes playgrounds:

- [Katacoda](#)
- [Play with Kubernetes](#)

To check the version, enter `kubectl version`.

## **Trade-offs**

The `kubectl` tool supports three kinds of object management:

- *Imperative commands*
- *Imperative object configuration*
- *Declarative object configuration*

See [Kubernetes Object Management](#) for a discussion of the advantages and disadvantage of each kind of object management.

## Overview

Declarative object configuration requires a firm understanding of the Kubernetes object definitions and configuration. Read and complete the following documents if you have not already:

- [Managing Kubernetes Objects Using Imperative Commands](#)
- [Imperative Management of Kubernetes Objects Using Configuration Files](#)

Following are definitions for terms used in this document:

- **object configuration file / configuration file:** A file that defines the configuration for a Kubernetes object. This topic shows how to pass configuration files to `kubectl apply`. Configuration files are typically stored in source control, such as Git.
- **live object configuration / live configuration:** The live configuration values of an object, as observed by the Kubernetes cluster. These are kept in the Kubernetes cluster storage, typically etcd.
- **declarative configuration writer / declarative writer:** A person or software component that makes updates to a live object. The live writers referred to in this topic make changes to object configuration files and run `kubectl apply` to write the changes.

## How to create objects

Use `kubectl apply` to create all objects, except those that already exist, defined by configuration files in a specified directory:

```
kubectl apply -f <directory>/
```

This sets the `kubectl.kubernetes.io/last-applied-configuration: '{...}'` annotation on each object. The annotation contains the contents of the object configuration file that was used to create the object.

**Note:** Add the `-R` flag to recursively process directories.

Here's an example of an object configuration file:

[application/simple\\_deployment.yaml](#)



```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  minReadySeconds: 5
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:1.14.2
          ports:
            - containerPort: 80
```

Run `kubectl diff` to print the object that will be created:

```
kubectl diff -f https://k8s.io/examples/application/
simple_deployment.yaml
```

**Note:**

`diff` uses [server-side dry-run](#), which needs to be enabled on `kube-apiserver`.

Since `diff` performs a server-side `apply` request in dry-run mode, it requires granting `PATCH`, `CREATE`, and `UPDATE` permissions. See [Dry-Run Authorization](#) for details.

Create the object using `kubectl apply`:

```
kubectl apply -f https://k8s.io/examples/application/
simple_deployment.yaml
```

Print the live configuration using `kubectl get`:

```
kubectl get -f https://k8s.io/examples/application/
simple_deployment.yaml -o yaml
```

The output shows that the `kubectl.kubernetes.io/last-applied-configuration` annotation was written to the live configuration, and it matches the configuration file:

```
kind: Deployment
metadata:
  annotations:
    # ...
    # This is the json representation of simple_deployment.yaml
    # It was written by kubectl apply when the object was created
    kubectl.kubernetes.io/last-applied-configuration: |
      {"apiVersion": "apps/v1", "kind": "Deployment",
       "metadata": {"annotations": {}, "name": "nginx-deployment", "namespace": "default"}, 
       "spec": {"minReadySeconds": 5, "selector": {"matchLabels": {"app": "nginx"}}, "template": {"metadata": {"labels": {"app": "nginx"}}, "spec": {"containers": [{"image": "nginx:1.14.2", "name": "nginx"}], "ports": [{"containerPort": 80}]}]}]}
    # ...
spec:
  # ...
  minReadySeconds: 5
  selector:
    matchLabels:
      # ...
      app: nginx
  template:
    metadata:
      # ...
      labels:
        app: nginx
  spec:
    containers:
      - image: nginx:1.14.2
        # ...
        name: nginx
        ports:
          - containerPort: 80
            # ...
      # ...
    # ...
  # ...
```

## How to update objects

You can also use `kubectl apply` to update all objects defined in a directory, even if those objects already exist. This approach accomplishes the following:

1. Sets fields that appear in the configuration file in the live configuration.
2. Clears fields removed from the configuration file in the live configuration.

```
kubectl diff -f <directory>/  
kubectl apply -f <directory>/
```

**Note:** Add the `-R` flag to recursively process directories.

Here's an example configuration file:

[application/simple\\_deployment.yaml](#)  


```
apiVersion: apps/v1  
kind: Deployment  
metadata:  
  name: nginx-deployment  
spec:  
  selector:  
    matchLabels:  
      app: nginx  
  minReadySeconds: 5  
  template:  
    metadata:  
      labels:  
        app: nginx  
    spec:  
      containers:  
      - name: nginx  
        image: nginx:1.14.2  
        ports:  
        - containerPort: 80
```

Create the object using `kubectl apply`:

```
kubectl apply -f https://k8s.io/examples/application/  
simple_deployment.yaml
```

**Note:** For purposes of illustration, the preceding command refers to a single configuration file instead of a directory.

Print the live configuration using `kubectl get`:

```
kubectl get -f https://k8s.io/examples/application/
simple_deployment.yaml -o yaml
```

The output shows that the `kubectl.kubernetes.io/last-applied-configuration` annotation was written to the live configuration, and it matches the configuration file:

```
kind: Deployment
metadata:
  annotations:
    # ...
    # This is the json representation of simple_deployment.yaml
    # It was written by kubectl apply when the object was created
  kubectl.kubernetes.io/last-applied-configuration: |
    {"apiVersion": "apps/v1", "kind": "Deployment",
     "metadata": {"annotations": {}, "name": "nginx-deployment", "namespace": "default"},
     "spec": {"minReadySeconds": 5, "selector": {"matchLabels": {"app": "nginx"}}, "template": {"metadata": {"labels": {"app": "nginx"}}, "spec": {"containers": [{"image": "nginx:1.14.2", "name": "nginx"}]}},
              "ports": [{"containerPort": 80}]}]}
  # ...
spec:
  # ...
  minReadySeconds: 5
  selector:
    matchLabels:
      # ...
      app: nginx
  template:
    metadata:
      # ...
      labels:
        app: nginx
  spec:
    containers:
      - image: nginx:1.14.2
        # ...
        name: nginx
        ports:
          - containerPort: 80
            # ...
            # ...
            # ...
    # ...
```

Directly update the `replicas` field in the live configuration by using `kubectl scale`. This does not use `kubectl apply`:

```
kubectl scale deployment/nginx-deployment --replicas=2
```

Print the live configuration using `kubectl get`:

```
kubectl get deployment nginx-deployment -o yaml
```

The output shows that the `replicas` field has been set to 2, and the `last-applied-configuration` annotation does not contain a `replicas` field:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  annotations:
    # ...
    # note that the annotation does not contain replicas
    # because it was not updated through apply
    kubectl.kubernetes.io/last-applied-configuration: |
      {"apiVersion": "apps/v1", "kind": "Deployment",
       "metadata": {"annotations": {}, "name": "nginx-deployment", "namespace": "default"},
       "spec": {"minReadySeconds": 5, "selector": {"matchLabels": {"app": "nginx"}}, "template": {"metadata": {"labels": {"app": "nginx"}}, "spec": {"containers": [{"image": "nginx:1.14.2", "name": "nginx"}]}},
      "ports": [{"containerPort": 80}]}]}
    # ...
spec:
  replicas: 2 # written by scale
  # ...
  minReadySeconds: 5
  selector:
    matchLabels:
      # ...
      app: nginx
  template:
    metadata:
      # ...
      labels:
        app: nginx
  spec:
    containers:
      - image: nginx:1.14.2
        # ...
        name: nginx
        ports:
          - containerPort: 80
    # ...
```

Update the `simple_deployment.yaml` configuration file to change the image from `nginx:1.14.2` to `nginx:1.16.1`, and delete the `minReadySeconds` field:

[application/update\\_deployment.yaml](#)  


```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
  spec:
    containers:
      - name: nginx
        image: nginx:1.16.1 # update the image
        ports:
          - containerPort: 80
```

Apply the changes made to the configuration file:

```
kubectl diff -f https://k8s.io/examples/application/
update_deployment.yaml
kubectl apply -f https://k8s.io/examples/application/
update_deployment.yaml
```

Print the live configuration using `kubectl get`:

```
kubectl get -f https://k8s.io/examples/application/
update_deployment.yaml -o yaml
```

The output shows the following changes to the live configuration:

- The `replicas` field retains the value of 2 set by `kubectl scale`. This is possible because it is omitted from the configuration file.
- The `image` field has been updated to `nginx:1.16.1` from `nginx:1.14.2`.
- The `last-applied-configuration` annotation has been updated with the new image.
- The `minReadySeconds` field has been cleared.
- The `last-applied-configuration` annotation no longer contains the `minReadySeconds` field.

```

apiVersion: apps/v1
kind: Deployment
metadata:
  annotations:
    # ...
    # The annotation contains the updated image to nginx 1.11.9,
    # but does not contain the updated replicas to 2
    kubectl.kubernetes.io/last-applied-configuration: |
      {"apiVersion":"apps/v1","kind":"Deployment",
       "metadata": {"annotations":{},"name":"nginx-deployment","namespace":"default"},
       "spec": {"selector": {"matchLabels":
        {"app": "nginx"}}, "template": {"metadata": {"labels": {"app": "nginx"}}, "spec": {"containers": [{"image": "nginx:1.16.1", "name": "nginx"}],
          "ports": [{"containerPort": 80}]}]}}
    # ...
spec:
  replicas: 2 # Set by `kubectl scale`. Ignored by `kubectl apply`.
  # minReadySeconds cleared by `kubectl apply`
  # ...
  selector:
    matchLabels:
      # ...
      app: nginx
  template:
    metadata:
      # ...
      labels:
        app: nginx
  spec:
    containers:
      - image: nginx:1.16.1 # Set by `kubectl apply`
        # ...
        name: nginx
        ports:
          - containerPort: 80
            # ...
        # ...
      # ...
    # ...
# ...

```

**Warning:** Mixing `kubectl apply` with the imperative object configuration commands `create` and `replace` is not supported. This is because `create` and `replace` do not retain the `kubectl.kubernetes.io/last-applied-configuration` that `kubectl apply` uses to compute updates.

# **How to delete objects**

*There are two approaches to delete objects managed by kubectl apply.*

## **Recommended: `kubectl delete -f <filename>`**

*Manually deleting objects using the imperative command is the recommended approach, as it is more explicit about what is being deleted, and less likely to result in the user deleting something unintentionally:*

```
kubectl delete -f <filename>
```

## **Alternative: `kubectl apply -f <directory/> --prune -l your=label`**

*Only use this if you know what you are doing.*

**Warning:** `kubectl apply --prune` is in alpha, and backwards incompatible changes might be introduced in subsequent releases.

**Warning:** You must be careful when using this command, so that you do not delete objects unintentionally.

*As an alternative to `kubectl delete`, you can use `kubectl apply` to identify objects to be deleted after their configuration files have been removed from the directory. Apply with `--prune` queries the API server for all objects matching a set of labels, and attempts to match the returned live object configurations against the object configuration files. If an object matches the query, and it does not have a configuration file in the directory, and it has a `last-applied-configuration` annotation, it is deleted.*

```
kubectl apply -f <directory/> --prune -l <labels>
```

**Warning:** Apply with prune should only be run against the root directory containing the object configuration files. Running against sub-directories can cause objects to be unintentionally deleted if they are returned by the label selector query specified with `-l <labels>` and do not appear in the subdirectory.

## **How to view an object**

You can use `kubectl get` with `-o yaml` to view the configuration of a live object:

```
kubectl get -f <filename/url> -o yaml
```

## **How apply calculates differences and merges changes**

**Caution:** A patch is an update operation that is scoped to specific fields of an object instead of the entire object. This enables updating only a specific set of fields on an object without reading the object first.

When `kubectl apply` updates the live configuration for an object, it does so by sending a patch request to the API server. The patch defines updates scoped to specific fields of the live object configuration. The `kubectl apply` command calculates this patch request using the configuration file, the live configuration, and the `last-applied-configuration` annotation stored in the live configuration.

### **Merge patch calculation**

The `kubectl apply` command writes the contents of the configuration file to the `kubectl.kubernetes.io/last-applied-configuration` annotation. This is used to identify fields that have been removed from the configuration file and need to be cleared from the live configuration. Here are the steps used to calculate which fields should be deleted or set:

1. Calculate the fields to delete. These are the fields present in `last-applied-configuration` and missing from the configuration file.
2. Calculate the fields to add or set. These are the fields present in the configuration file whose values don't match the live configuration.

Here's an example. Suppose this is the configuration file for a Deployment object:

[application/update\\_deployment.yaml](#)  


```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
  spec:
    containers:
      - name: nginx
        image: nginx:1.16.1 # update the image
        ports:
          - containerPort: 80

```

Also, suppose this is the live configuration for the same Deployment object:

```

apiVersion: apps/v1
kind: Deployment
metadata:
  annotations:
    # ...
    # note that the annotation does not contain replicas
    # because it was not updated through apply
    kubectl.kubernetes.io/last-applied-configuration: |
      {"apiVersion":"apps/v1","kind":"Deployment",
       "metadata": {"annotations":{}, "name": "nginx-deployment", "namespace": "default"},
       "spec": {"minReadySeconds":5, "selector": {"matchLabels": {"app": "nginx"}}, "template": {"metadata": {"labels": {"app": "nginx"}}, "spec": {"containers": [{"image": "nginx:1.14.2", "name": "nginx"}], "ports": [{"containerPort": 80}]}]}]}
    # ...
spec:
  replicas: 2 # written by scale
  # ...
  minReadySeconds: 5
  selector:
    matchLabels:
      # ...
      app: nginx
  template:
    metadata:
      # ...
      labels:

```

```

    app: nginx
spec:
  containers:
    - image: nginx:1.14.2
      # ...
      name: nginx
      ports:
        - containerPort: 80
    # ...

```

Here are the merge calculations that would be performed by `kubectl apply`:

1. Calculate the fields to delete by reading values from `last-applied-configuration` and comparing them to values in the configuration file. Clear fields explicitly set to null in the local object configuration file regardless of whether they appear in the `last-applied-configuration`. In this example, `minReadySeconds` appears in the `last-applied-configuration` annotation, but does not appear in the configuration file. **Action:** Clear `minReadySeconds` from the live configuration.
2. Calculate the fields to set by reading values from the configuration file and comparing them to values in the live configuration. In this example, the value of `image` in the configuration file does not match the value in the live configuration. **Action:** Set the value of `image` in the live configuration.
3. Set the `last-applied-configuration` annotation to match the value of the configuration file.
4. Merge the results from 1, 2, 3 into a single patch request to the API server.

Here is the live configuration that is the result of the merge:

```

apiVersion: apps/v1
kind: Deployment
metadata:
  annotations:
    # ...
    # The annotation contains the updated image to nginx 1.11.9,
    # but does not contain the updated replicas to 2
    kubectl.kubernetes.io/last-applied-configuration: |
      {"apiVersion": "apps/v1", "kind": "Deployment",
       "metadata": {"annotations": {}, "name": "nginx-deployment", "namespace": "default"}, 
       "spec": {"selector": {"matchLabels": {"app": "nginx"}}, "template": {"metadata": {"labels": {"app": "nginx"}}, "spec": {"containers": [{"image": "nginx:1.16.1", "name": "nginx"}], "ports": [{"containerPort": 80}]}}}}

```

```

# ...
spec:
  selector:
    matchLabels:
      # ...
      app: nginx
  replicas: 2 # Set by `kubectl scale`. Ignored by `kubectl apply`.
  # minReadySeconds cleared by `kubectl apply`
  # ...
  template:
    metadata:
      # ...
      labels:
        app: nginx
  spec:
    containers:
      - image: nginx:1.16.1 # Set by `kubectl apply`
        # ...
        name: nginx
        ports:
          - containerPort: 80
        # ...
        # ...
      # ...
    # ...
# ...

```

## **How different types of fields are merged**

*How a particular field in a configuration file is merged with the live configuration depends on the type of the field. There are several types of fields:*

- **primitive:** A field of type *string*, *integer*, or *boolean*. For example, *image* and *replicas* are primitive fields. **Action:** Replace.
- **map, also called object:** A field of type *map* or a complex type that contains subfields. For example, *labels*, *annotations*, *spec* and *metadata* are all maps. **Action:** Merge elements or subfields.
- **list:** A field containing a list of items that can be either primitive types or maps. For example, *containers*, *ports*, and *args* are lists. **Action:** Varies.

*When kubectl apply updates a map or list field, it typically does not replace the entire field, but instead updates the individual subelements. For instance, when merging the spec on a Deployment, the entire spec is not replaced. Instead the subfields of spec, such as replicas, are compared and merged.*

## **Merging changes to primitive fields**

Primitive fields are replaced or cleared.

**Note:** - is used for "not applicable" because the value is not used.

<b>Field in object configuration file</b>	<b>Field in live object configuration</b>	<b>Field in last-applied-configuration</b>	<b>Action</b>
Yes	Yes	-	Set live to configuration file value.
Yes	No	-	Set live to local configuration.
No	-	Yes	Clear from live configuration.
No	-	No	Do nothing. Keep live value.

## **Merging changes to map fields**

Fields that represent maps are merged by comparing each of the subfields or elements of the map:

**Note:** - is used for "not applicable" because the value is not used.

<b>Key in object configuration file</b>	<b>Key in live object configuration</b>	<b>Field in last-applied-configuration</b>	<b>Action</b>
Yes	Yes	-	Compare sub fields values.
Yes	No	-	Set live to local configuration.
No	-	Yes	Delete from live configuration.
No	-	No	Do nothing. Keep live value.

## **Merging changes for fields of type list**

Merging changes to a list uses one of three strategies:

- Replace the list if all its elements are primitives.
- Merge individual elements in a list of complex elements.

- Merge a list of primitive elements.

*The choice of strategy is made on a per-field basis.*

### **Replace the list if all its elements are primitives**

*Treat the list the same as a primitive field. Replace or delete the entire list. This preserves ordering.*

**Example:** Use `kubectl apply` to update the `args` field of a Container in a Pod. This sets the value of `args` in the live configuration to the value in the configuration file. Any `args` elements that had previously been added to the live configuration are lost. The order of the `args` elements defined in the configuration file is retained in the live configuration.

```
# last-applied-configuration value
  args: ["a", "b"]

# configuration file value
  args: ["a", "c"]

# live configuration
  args: ["a", "b", "d"]

# result after merge
  args: ["a", "c"]
```

**Explanation:** The merge used the configuration file value as the new list value.

### **Merge individual elements of a list of complex elements:**

*Treat the list as a map, and treat a specific field of each element as a key. Add, delete, or update individual elements. This does not preserve ordering.*

*This merge strategy uses a special tag on each field called a `patchMergeKey`. The `patchMergeKey` is defined for each field in the Kubernetes source code: [types.go](#) When merging a list of maps, the field specified as the `patchMergeKey` for a given element is used like a map key for that element.*

**Example:** Use `kubectl apply` to update the `containers` field of a `PodSpec`. This merges the list as though it was a map where each element is keyed by name.

```

# last-applied-configuration value
  containers:
    - name: nginx
      image: nginx:1.16
    - name: nginx-helper-a # key: nginx-helper-a; will be
      deleted in result
      image: helper:1.3
    - name: nginx-helper-b # key: nginx-helper-b; will be retained
      image: helper:1.3

# configuration file value
  containers:
    - name: nginx
      image: nginx:1.16
    - name: nginx-helper-b
      image: helper:1.3
    - name: nginx-helper-c # key: nginx-helper-c; will be added in
      result
      image: helper:1.3

# live configuration
  containers:
    - name: nginx
      image: nginx:1.16
    - name: nginx-helper-a
      image: helper:1.3
    - name: nginx-helper-b
      image: helper:1.3
      args: ["run"] # Field will be retained
    - name: nginx-helper-d # key: nginx-helper-d; will be retained
      image: helper:1.3

# result after merge
  containers:
    - name: nginx
      image: nginx:1.16
      # Element nginx-helper-a was deleted
    - name: nginx-helper-b
      image: helper:1.3
      args: ["run"] # Field was retained
    - name: nginx-helper-c # Element was added
      image: helper:1.3
    - name: nginx-helper-d # Element was ignored
      image: helper:1.3

```

**Explanation:**

- The container named "nginx-helper-a" was deleted because no container named "nginx-helper-a" appeared in the configuration file.
- The container named "nginx-helper-b" retained the changes to args in the live configuration. kubectl apply was able to identify that "nginx-helper-b" in the live configuration was the same "nginx-helper-b" as in the configuration file, even though their fields had different values (no args in the configuration file). This is because the patchMergeKey field value (name) was identical in both.
- The container named "nginx-helper-c" was added because no container with that name appeared in the live configuration, but one with that name appeared in the configuration file.
- The container named "nginx-helper-d" was retained because no element with that name appeared in the last-applied-configuration.

## Merge a list of primitive elements

As of Kubernetes 1.5, merging lists of primitive elements is not supported.

**Note:** Which of the above strategies is chosen for a given field is controlled by the patchStrategy tag in [types.go](#). If no patchStrategy is specified for a field of type list, then the list is replaced.

## Default field values

The API server sets certain fields to default values in the live configuration if they are not specified when the object is created.

Here's a configuration file for a Deployment. The file does not specify strategy:

[application/simple\\_deployment.yaml](#)  
□

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  minReadySeconds: 5
  template:
    metadata:
      labels:
```

```
    app: nginx
  spec:
    containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
          - containerPort: 80
```

Create the object using `kubectl apply`:

```
kubectl apply -f https://k8s.io/examples/application/
simple_deployment.yaml
```

Print the live configuration using `kubectl get`:

```
kubectl get -f https://k8s.io/examples/application/
simple_deployment.yaml -o yaml
```

The output shows that the API server set several fields to default values in the live configuration. These fields were not specified in the configuration file.

```
apiVersion: apps/v1
kind: Deployment
# ...
spec:
  selector:
    matchLabels:
      app: nginx
  minReadySeconds: 5
  replicas: 1 # defaulted by apiserver
  strategy:
    rollingUpdate: # defaulted by apiserver - derived from
strategy.type
    maxSurge: 1
    maxUnavailable: 1
    type: RollingUpdate # defaulted by apiserver
  template:
    metadata:
      creationTimestamp: null
    labels:
      app: nginx
  spec:
    containers:
      - image: nginx:1.14.2
        imagePullPolicy: IfNotPresent # defaulted by apiserver
        name: nginx
```

```

ports:
  - containerPort: 80
    protocol: TCP # defaulted by apiserver
  resources: {} # defaulted by apiserver
  terminationMessagePath: /dev/termination-log # defaulted
by apiserver
  dnsPolicy: ClusterFirst # defaulted by apiserver
  restartPolicy: Always # defaulted by apiserver
  securityContext: {} # defaulted by apiserver
  terminationGracePeriodSeconds: 30 # defaulted by apiserver
# ...

```

*In a patch request, defaulted fields are not re-defaulted unless they are explicitly cleared as part of a patch request. This can cause unexpected behavior for fields that are defaulted based on the values of other fields. When the other fields are later changed, the values defaulted from them will not be updated unless they are explicitly cleared.*

*For this reason, it is recommended that certain fields defaulted by the server are explicitly defined in the configuration file, even if the desired values match the server defaults. This makes it easier to recognize conflicting values that will not be re-defaulted by the server.*

### **Example:**

```

# last-applied-configuration
spec:
  template:
    metadata:
      labels:
        app: nginx
  spec:
    containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
          - containerPort: 80

# configuration file
spec:
  strategy:
    type: Recreate # updated value
  template:
    metadata:
      labels:
        app: nginx
  spec:
    containers:
      - name: nginx

```

```

  image: nginx:1.14.2
  ports:
    - containerPort: 80

# live configuration
spec:
  strategy:
    type: RollingUpdate # defaulted value
    rollingUpdate: # defaulted value derived from type
      maxSurge : 1
      maxUnavailable: 1
  template:
    metadata:
      labels:
        app: nginx
  spec:
    containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
          - containerPort: 80

# result after merge - ERROR!
spec:
  strategy:
    type: Recreate # updated value: incompatible with rollingUpdate
    rollingUpdate: # defaulted value: incompatible with "type: Recreate"
      maxSurge : 1
      maxUnavailable: 1
  template:
    metadata:
      labels:
        app: nginx
  spec:
    containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
          - containerPort: 80

```

### **Explanation:**

1. The user creates a Deployment without defining `strategy.type`.
2. The server defaults `strategy.type` to `RollingUpdate` and defaults the `strategy.rollingUpdate` values.
3. The user changes `strategy.type` to `Recreate`. The `strategy.rollingUpdate` values remain at their defaulted values, though the server expects them to be cleared. If the `strategy.rollingUpdate` had

*been defined initially in the configuration file, it would have been more clear that they needed to be deleted.*

4. *Apply fails because strategy.rollingUpdate is not cleared. The strategy.rollingupdate field cannot be defined with a strategy.type of Recreate.*

*Recommendation:* These fields should be explicitly defined in the object configuration file:

- *Selectors and PodTemplate labels on workloads, such as Deployment, StatefulSet, Job, DaemonSet, ReplicaSet, and ReplicationController*
- *Deployment rollout strategy*

## **How to clear server-defaulted fields or fields set by other writers**

*Fields that do not appear in the configuration file can be cleared by setting their values to null and then applying the configuration file. For fields defaulted by the server, this triggers re-defaulting the values.*

## **How to change ownership of a field between the configuration file and direct imperative writers**

*These are the only methods you should use to change an individual object field:*

- *Use kubectl apply.*
- *Write directly to the live configuration without modifying the configuration file: for example, use kubectl scale.*

## **Changing the owner from a direct imperative writer to a configuration file**

*Add the field to the configuration file. For the field, discontinue direct updates to the live configuration that do not go through kubectl apply.*

## **Changing the owner from a configuration file to a direct imperative writer**

*As of Kubernetes 1.5, changing ownership of a field from a configuration file to an imperative writer requires manual steps:*

- Remove the field from the configuration file.
- Remove the field from the `kubectl.kubernetes.io/last-applied-configuration` annotation on the live object.

## ***Changing management methods***

*Kubernetes objects should be managed using only one method at a time. Switching from one method to another is possible, but is a manual process.*

**Note:** It is OK to use imperative deletion with declarative management.

### ***Migrating from imperative command management to declarative object configuration***

*Migrating from imperative command management to declarative object configuration involves several manual steps:*

1. Export the live object to a local configuration file:

```
kubectl get <kind>/<name> -o yaml > <kind>_<name>.yaml
```

2. Manually remove the `status` field from the configuration file.

**Note:** This step is optional, as `kubectl apply` does not update the `status` field even if it is present in the configuration file.

3. Set the `kubectl.kubernetes.io/last-applied-configuration` annotation on the object:

```
kubectl replace --save-config -f <kind>_<name>.yaml
```

4. Change processes to use `kubectl apply` for managing the object exclusively.

### ***Migrating from imperative object configuration to declarative object configuration***

1. Set the `kubectl.kubernetes.io/last-applied-configuration` annotation on the object:

```
kubectl replace --save-config -f <kind>_<name>.yaml
```

2. Change processes to use `kubectl apply` for managing the object exclusively.

## Defining controller selectors and PodTemplate labels

**Warning:** Updating selectors on controllers is strongly discouraged.

The recommended approach is to define a single, immutable PodTemplate label used only by the controller selector with no other semantic meaning.

**Example:**

```
selector:  
  matchLabels:  
    controller-selector: "apps/v1/deployment/nginx"  
template:  
  metadata:  
    labels:  
      controller-selector: "apps/v1/deployment/nginx"
```

## What's next

- [Managing Kubernetes Objects Using Imperative Commands](#)
- [Imperative Management of Kubernetes Objects Using Configuration Files](#)
- [Kubectl Command Reference](#)
- [Kubernetes API 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](#).

Last modified August 07, 2020 at 4:46 PM PST: [Tune links in tasks section \(1/2\) \(b8541d212\)](#)

[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Trade-offs](#)
- [Overview](#)
- [How to create objects](#)
- [How to update objects](#)
- [How to delete objects](#)
  - [Recommended: kubectl delete -f <filename>](#)
  - [Alternative: kubectl apply -f <directory/> --prune -l your=label](#)
- [How to view an object](#)
- [How apply calculates differences and merges changes](#)
  - [Merge patch calculation](#)
  - [How different types of fields are merged](#)
  - [Merging changes to primitive fields](#)
  - [Merging changes to map fields](#)
  - [Merging changes for fields of type list](#)
- [Default field values](#)
  - [How to clear server-defaulted fields or fields set by other writers](#)
- [How to change ownership of a field between the configuration file and direct imperative writers](#)
  - [Changing the owner from a direct imperative writer to a configuration file](#)
  - [Changing the owner from a configuration file to a direct imperative writer](#)
- [Changing management methods](#)
  - [Migrating from imperative command management to declarative object configuration](#)
  - [Migrating from imperative object configuration to declarative object configuration](#)
- [Defining controller selectors and PodTemplate labels](#)
- [What's next](#)

# Declarative Management of Kubernetes Objects Using Kustomize

[Kustomize](#) is a standalone tool to customize Kubernetes objects through a [kustomization file](#).

Since 1.14, Kubectl also supports the management of Kubernetes objects using a kustomization file. To view Resources found in a directory containing a kustomization file, run the following command:

```
kubectl kustomize <kustomization_directory>
```

To apply those Resources, run `kubectl apply` with `--kustomize` or `-k` flag:

```
kubectl apply -k <kustomization_directory>
```

## Before you begin

Install [kubectl](#).

You need to have a Kubernetes cluster, and the `kubectl` command-line tool must be configured to communicate with your cluster. If you do not already have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:

- [Katacoda](#)
- [Play with Kubernetes](#)

To check the version, enter `kubectl version`.

## Overview of Kustomize

Kustomize is a tool for customizing Kubernetes configurations. It has the following features to manage application configuration files:

- generating resources from other sources
- setting cross-cutting fields for resources
- composing and customizing collections of resources

## Generating Resources

ConfigMaps and Secrets hold configuration or sensitive data that are used by other Kubernetes objects, such as Pods. The source of truth of ConfigMaps or Secrets are usually external to a cluster, such as a `.properties` file or an SSH keyfile. Kustomize has `secretGenerator` and `configMapGenerator`, which generate Secret and ConfigMap from files or literals.

### `configMapGenerator`

To generate a ConfigMap from a file, add an entry to the `files` list in `configMapGenerator`. Here is an example of generating a ConfigMap with a data item from a `.properties` file:

```
# Create a application.properties file
cat <<EOF >application.properties
FOO=Bar
EOF
```

```
cat <<EOF >./kustomization.yaml
configMapGenerator:
- name: example-configmap-1
  files:
  - application.properties
EOF
```

The generated ConfigMap can be examined with the following command:

```
kubectl kustomize ./
```

The generated ConfigMap is:

```
apiVersion: v1
data:
  application.properties: |
    FOO=Bar
kind: ConfigMap
metadata:
  name: example-configmap-1-8mbdf7882g
```

ConfigMaps can also be generated from literal key-value pairs. To generate a ConfigMap from a literal key-value pair, add an entry to the `literals` list in `configMapGenerator`. Here is an example of generating a ConfigMap with a data item from a key-value pair:

```
cat <<EOF >./kustomization.yaml
configMapGenerator:
- name: example-configmap-2
  literals:
  - FOO=Bar
EOF
```

The generated ConfigMap can be checked by the following command:

```
kubectl kustomize ./
```

The generated ConfigMap is:

```
apiVersion: v1
data:
  FOO: Bar
kind: ConfigMap
```

```
metadata:
  name: example-configmap-2-g2hdhfc6tk
```

### **secretGenerator**

You can generate Secrets from files or literal key-value pairs. To generate a Secret from a file, add an entry to the `files` list in `secretGenerator`. Here is an example of generating a Secret with a data item from a file:

```
# Create a password.txt file
cat <<EOF >./password.txt
username=admin
password=secret
EOF

cat <<EOF >./kustomization.yaml
secretGenerator:
- name: example-secret-1
  files:
  - password.txt
EOF
```

The generated Secret is as follows:

```
apiVersion: v1
data:
  password.txt: dXNlcm5hbWU9YWRtaW4KcGFzc3dvcmQ9c2VjcmV0Cg==
kind: Secret
metadata:
  name: example-secret-1-t2kt65hgtb
type: Opaque
```

To generate a Secret from a literal key-value pair, add an entry to `literals` list in `secretGenerator`. Here is an example of generating a Secret with a data item from a key-value pair:

```
cat <<EOF >./kustomization.yaml
secretGenerator:
- name: example-secret-2
  literals:
  - username=admin
  - password=secret
EOF
```

The generated Secret is as follows:

```
apiVersion: v1
data:
  password: c2VjcmV0
  username: YwRtaW4=
kind: Secret
metadata:
  name: example-secret-2-t52t6g96d8
type: Opaque
```

## generatorOptions

The generated ConfigMaps and Secrets have a content hash suffix appended. This ensures that a new ConfigMap or Secret is generated when the contents are changed. To disable the behavior of appending a suffix, one can use generatorOptions. Besides that, it is also possible to specify cross-cutting options for generated ConfigMaps and Secrets.

```
cat <<EOF >./kustomization.yaml
configMapGenerator:
- name: example-configmap-3
  literals:
  - FOO=Bar
generatorOptions:
  disableNameSuffixHash: true
  labels:
    type: generated
  annotations:
    note: generated
EOF
```

Run `kubectl kustomize ./` to view the generated ConfigMap:

```
apiVersion: v1
data:
  FOO: Bar
kind: ConfigMap
metadata:
  annotations:
    note: generated
  labels:
    type: generated
  name: example-configmap-3
```

## Setting cross-cutting fields

*It is quite common to set cross-cutting fields for all Kubernetes resources in a project. Some use cases for setting cross-cutting fields:*

- *setting the same namespace for all Resources*
- *adding the same name prefix or suffix*
- *adding the same set of labels*
- *adding the same set of annotations*

*Here is an example:*

```
# Create a deployment.yaml
cat <<EOF >./deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx
EOF

cat <<EOF >./kustomization.yaml
namespace: my-namespace
namePrefix: dev-
nameSuffix: "-001"
commonLabels:
  app: bingo
commonAnnotations:
  oncallPager: 800-555-1212
resources:
- deployment.yaml
EOF
```

*Run `kubectl kustomize ./` to view those fields are all set in the Deployment Resource:*

```

apiVersion: apps/v1
kind: Deployment
metadata:
  annotations:
    oncallPager: 800-555-1212
  labels:
    app: bingo
  name: dev-nginx-deployment-001
  namespace: my-namespace
spec:
  selector:
    matchLabels:
      app: bingo
  template:
    metadata:
      annotations:
        oncallPager: 800-555-1212
      labels:
        app: bingo
    spec:
      containers:
        - image: nginx
          name: nginx

```

## **Composing and Customizing Resources**

*It is common to compose a set of Resources in a project and manage them inside the same file or directory. Kustomize offers composing Resources from different files and applying patches or other customization to them.*

### **Composing**

*Kustomize supports composition of different resources. The `resources` field, in the `kustomization.yaml` file, defines the list of resources to include in a configuration. Set the path to a resource's configuration file in the `resources` list. Here is an example of an NGINX application comprised of a Deployment and a Service:*

```

# Create a deployment.yaml file
cat <<EOF > deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-nginx
spec:
  selector:
    matchLabels:

```

```

    run: my-nginx
replicas: 2
template:
  metadata:
    labels:
      run: my-nginx
spec:
  containers:
    - name: my-nginx
      image: nginx
      ports:
        - containerPort: 80
EOF

# Create a service.yaml file
cat <<EOF > service.yaml
apiVersion: v1
kind: Service
metadata:
  name: my-nginx
  labels:
    run: my-nginx
spec:
  ports:
    - port: 80
      protocol: TCP
  selector:
    run: my-nginx
EOF

# Create a kustomization.yaml composing them
cat <<EOF >./kustomization.yaml
resources:
- deployment.yaml
- service.yaml
EOF

```

*The Resources from `kubectl kustomize ./` contain both the Deployment and the Service objects.*

## **Customizing**

Patches can be used to apply different customizations to Resources. Kustomize supports different patching mechanisms through `patchesStrategicMerge` and `patchesJson6902`. `patchesStrategicMerge` is a list of file paths. Each file should be resolved to a [strategic merge patch](#). The names inside the patches must match Resource names that are already loaded. Small patches that do one thing are recommended. For example, create one

*patch for increasing the deployment replica number and another patch for setting the memory limit.*

```
# Create a deployment.yaml file
cat <<EOF > deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-nginx
spec:
  selector:
    matchLabels:
      run: my-nginx
  replicas: 2
  template:
    metadata:
      labels:
        run: my-nginx
    spec:
      containers:
        - name: my-nginx
          image: nginx
          ports:
            - containerPort: 80
EOF

# Create a patch increase_replicas.yaml
cat <<EOF > increase_replicas.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-nginx
spec:
  replicas: 3
EOF

# Create another patch set_memory.yaml
cat <<EOF > set_memory.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-nginx
spec:
  template:
    spec:
      containers:
        - name: my-nginx
          resources:
            limits:
              memory: 512Mi
EOF
```

```
cat <<EOF >./kustomization.yaml
resources:
- deployment.yaml
patchesStrategicMerge:
- increase_replicas.yaml
- set_memory.yaml
EOF
```

Run `kubectl kustomize ./` to view the Deployment:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      run: my-nginx
  template:
    metadata:
      labels:
        run: my-nginx
    spec:
      containers:
      - image: nginx
        limits:
          memory: 512Mi
        name: my-nginx
        ports:
        - containerPort: 80
```

Not all Resources or fields support strategic merge patches. To support modifying arbitrary fields in arbitrary Resources, Kustomize offers applying [JSON patch](#) through `patchesJson6902`. To find the correct Resource for a Json patch, the group, version, kind and name of that Resource need to be specified in `kustomization.yaml`. For example, increasing the replica number of a Deployment object can also be done through `patchesJson6902`.

```
# Create a deployment.yaml file
cat <<EOF > deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-nginx
spec:
  selector:
    matchLabels:
```

```

    run: my-nginx
replicas: 2
template:
  metadata:
    labels:
      run: my-nginx
spec:
  containers:
  - name: my-nginx
    image: nginx
    ports:
    - containerPort: 80
EOF

# Create a json patch
cat <<EOF > patch.yaml
- op: replace
  path: /spec/relicas
  value: 3
EOF

# Create a kustomization.yaml
cat <<EOF >./kustomization.yaml
resources:
- deployment.yaml

patchesJson6902:
- target:
    group: apps
    version: v1
    kind: Deployment
    name: my-nginx
    path: patch.yaml
EOF

```

Run `kubectl kustomize .` to see the `replicas` field is updated:

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      run: my-nginx
  template:
    metadata:
      labels:
        run: my-nginx

```

```
spec:  
  containers:  
    - image: nginx  
      name: my-nginx  
      ports:  
        - containerPort: 80
```

In addition to patches, Kustomize also offers customizing container images or injecting field values from other objects into containers without creating patches. For example, you can change the image used inside containers by specifying the new image in `images` field in `kustomization.yaml`.

```
cat <<EOF > deployment.yaml  
apiVersion: apps/v1  
kind: Deployment  
metadata:  
  name: my-nginx  
spec:  
  selector:  
    matchLabels:  
      run: my-nginx  
  replicas: 2  
  template:  
    metadata:  
      labels:  
        run: my-nginx  
    spec:  
      containers:  
        - name: my-nginx  
          image: nginx  
          ports:  
            - containerPort: 80  
EOF
```

```
cat <<EOF >./kustomization.yaml  
resources:  
- deployment.yaml  
images:  
- name: nginx  
  newName: my.image.registry/nginx  
  newTag: 1.4.0  
EOF
```

Run `kubectl kustomize ./` to see that the image being used is updated:

```
apiVersion: apps/v1  
kind: Deployment  
metadata:  
  name: my-nginx
```

```

spec:
  replicas: 2
  selector:
    matchLabels:
      run: my-nginx
  template:
    metadata:
      labels:
        run: my-nginx
  spec:
    containers:
      - image: my.image.registry/nginx:1.4.0
        name: my-nginx
        ports:
          - containerPort: 80

```

*Sometimes, the application running in a Pod may need to use configuration values from other objects. For example, a Pod from a Deployment object need to read the corresponding Service name from Env or as a command argument. Since the Service name may change as namePrefix or nameSuffix is added in the kustomization.yaml file. It is not recommended to hard code the Service name in the command argument. For this usage, Kustomize can inject the Service name into containers through vars.*

```

# Create a deployment.yaml file
cat <<EOF > deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-nginx
spec:
  selector:
    matchLabels:
      run: my-nginx
  replicas: 2
  template:
    metadata:
      labels:
        run: my-nginx
  spec:
    containers:
      - name: my-nginx
        image: nginx
        command: ["start", "--host", "$(MY_SERVICE_NAME)"]
EOF

# Create a service.yaml file
cat <<EOF > service.yaml
apiVersion: v1
kind: Service

```

```

metadata:
  name: my-nginx
  labels:
    run: my-nginx
spec:
  ports:
  - port: 80
    protocol: TCP
  selector:
    run: my-nginx
EOF

cat <<EOF >./kustomization.yaml
namePrefix: dev-
nameSuffix: "-001"

resources:
- deployment.yaml
- service.yaml

vars:
- name: MY_SERVICE_NAME
  objref:
    kind: Service
    name: my-nginx
    apiVersion: v1
EOF

```

Run `kubectl kustomize ./` to see that the Service name injected into containers is `dev-my-nginx-001`:

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: dev-my-nginx-001
spec:
  replicas: 2
  selector:
    matchLabels:
      run: my-nginx
  template:
    metadata:
      labels:
        run: my-nginx
  spec:
    containers:
    - command:
      - start
      - --host
      - dev-my-nginx-001

```

```
image: nginx
name: my-nginx
```

## Bases and Overlays

Kustomize has the concepts of **bases** and **overlays**. A **base** is a directory with a `kustomization.yaml`, which contains a set of resources and associated customization. A base could be either a local directory or a directory from a remote repo, as long as a `kustomization.yaml` is present inside. An **overlay** is a directory with a `kustomization.yaml` that refers to other kustomization directories as its bases. A **base** has no knowledge of an overlay and can be used in multiple overlays. An overlay may have multiple bases and it composes all resources from bases and may also have customization on top of them.

Here is an example of a base:

```
# Create a directory to hold the base
mkdir base
# Create a base/deployment.yaml
cat <<EOF > base/deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-nginx
spec:
  selector:
    matchLabels:
      run: my-nginx
  replicas: 2
  template:
    metadata:
      labels:
        run: my-nginx
    spec:
      containers:
        - name: my-nginx
          image: nginx
EOF

# Create a base/service.yaml file
cat <<EOF > base/service.yaml
apiVersion: v1
kind: Service
metadata:
  name: my-nginx
  labels:
    run: my-nginx
```

```

spec:
  ports:
    - port: 80
      protocol: TCP
  selector:
    run: my-nginx
EOF
# Create a base/kustomization.yaml
cat <<EOF > base/kustomization.yaml
resources:
- deployment.yaml
- service.yaml
EOF

```

*This base can be used in multiple overlays. You can add different namePrefix or other cross-cutting fields in different overlays. Here are two overlays using the same base.*

```

mkdir dev
cat <<EOF > dev/kustomization.yaml
bases:
- ./base
namePrefix: dev-
EOF

mkdir prod
cat <<EOF > prod/kustomization.yaml
bases:
- ./base
namePrefix: prod-
EOF

```

## **How to apply/view/delete objects using Kustomize**

*Use --kustomize or -k in kubectl commands to recognize Resources managed by kustomization.yaml. Note that -k should point to a kustomization directory, such as*

```
kubectl apply -k <kustomization directory>/
```

*Given the following kustomization.yaml,*

```
# Create a deployment.yaml file
cat <<EOF > deployment.yaml
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-nginx
spec:
  selector:
    matchLabels:
      run: my-nginx
  replicas: 2
  template:
    metadata:
      labels:
        run: my-nginx
    spec:
      containers:
        - name: my-nginx
          image: nginx
          ports:
            - containerPort: 80
EOF

# Create a kustomization.yaml
cat <<EOF >./kustomization.yaml
namePrefix: dev-
commonLabels:
  app: my-nginx
resources:
- deployment.yaml
EOF
```

Run the following command to apply the Deployment object `dev-my-nginx`:

```
> kubectl apply -k ./  
deployment.apps/dev-my-nginx created
```

Run one of the following commands to view the Deployment object `dev-my-nginx`:

```
kubectl get -k ./
```

```
kubectl describe -k ./
```

Run the following command to compare the Deployment object `dev-my-nginx` against the state that the cluster would be in if the manifest was applied:

```
kubectl diff -k ./
```

Run the following command to delete the Deployment object `dev-my-nginx`:

```
> kubectl delete -k ./  
deployment.apps "dev-my-nginx" deleted
```

## Kustomize Feature List

Field	Type	Explanation
<code>namespace</code>	<code>string</code>	<i>add namespace to all resources</i>
<code>namePrefix</code>	<code>string</code>	<i>value of this field is prepended to the names of all resources</i>
<code>nameSuffix</code>	<code>string</code>	<i>value of this field is appended to the names of all resources</i>
<code>commonLabels</code>	<code>map[string]string</code>	<i>labels to add to all resources and selectors</i>
<code>commonAnnotations</code>	<code>map[string]string</code>	<i>annotations to add to all resources</i>
<code>resources</code>	<code>[]string</code>	<i>each entry in this list must resolve to an existing resource configuration file</i>
<code>configmapGenerator</code>	<code>[]ConfigMapArgs</code>	<i>Each entry in this list generates a ConfigMap</i>
<code>secretGenerator</code>	<code>[]SecretArgs</code>	<i>Each entry in this list generates a Secret</i>
<code>generatorOptions</code>	<code>GeneratorOptions</code>	<i>Modify behaviors of all ConfigMap and Secret generator</i>
<code>bases</code>	<code>[]string</code>	<i>Each entry in this list should resolve to a directory containing a <code>kustomization.yaml</code> file</i>
<code>patchesStrategicMerge</code>	<code>[]string</code>	<i>Each entry in this list should resolve a strategic merge patch of a Kubernetes object</i>
<code>patchesJson6902</code>	<code>[]Json6902</code>	<i>Each entry in this list should resolve to a Kubernetes object and a Json Patch</i>
<code>vars</code>	<code>[]Var</code>	<i>Each entry is to capture text from one resource's field</i>
<code>images</code>	<code>[]Image</code>	<i>Each entry is to modify the name, tags and/or digest for one image without creating patches</i>
<code>configurations</code>	<code>[]string</code>	<i>Each entry in this list should resolve to a file containing <a href="#">Kustomize transformer configurations</a></i>

<b>Field</b>	<b>Type</b>	<b>Explanation</b>
crds	[]string	<i>Each entry in this list should resolve to an OpenAPI definition file for Kubernetes types</i>

## What's next

- [Kustomize](#)
- [Kubectl Book](#)
- [Kubectl Command Reference](#)
- [Kubernetes API 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](#).

Last modified December 11, 2020 at 9:12 PM PST: [Update kustomization.md \(7fdc4b6ed\)](#)

[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Overview of Kustomize](#)
  - [Generating Resources](#)
  - [Setting cross-cutting fields](#)
  - [Composing and Customizing Resources](#)
- [Bases and Overlays](#)
- [How to apply/view/delete objects using Kustomize](#)
- [Kustomize Feature List](#)
- [What's next](#)

# Managing Kubernetes Objects Using Imperative Commands

Kubernetes objects can quickly be created, updated, and deleted directly using imperative commands built into the `kubectl` command-line tool. This document explains how those commands are organized and how to use them to manage live objects.

# **Before you begin**

Install [kubectl](#).

You need to have a Kubernetes cluster, and the `kubectl` command-line tool must be configured to communicate with your cluster. If you do not already have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:

- [Katacoda](#)
- [Play with Kubernetes](#)

To check the version, enter `kubectl version`.

## **Trade-offs**

The `kubectl` tool supports three kinds of object management:

- Imperative commands
- Imperative object configuration
- Declarative object configuration

See [Kubernetes Object Management](#) for a discussion of the advantages and disadvantage of each kind of object management.

## **How to create objects**

The `kubectl` tool supports verb-driven commands for creating some of the most common object types. The commands are named to be recognizable to users unfamiliar with the Kubernetes object types.

- `run`: Create a new Pod to run a Container.
- `expose`: Create a new Service object to load balance traffic across Pods.
- `autoscale`: Create a new Autoscaler object to automatically horizontally scale a controller, such as a Deployment.

The `kubectl` tool also supports creation commands driven by object type. These commands support more object types and are more explicit about their intent, but require users to know the type of objects they intend to create.

- `create <objecttype> [<subtype>] <instancename>`

*Some objects types have subtypes that you can specify in the `create` command. For example, the `Service` object has several subtypes including `ClusterIP`, `LoadBalancer`, and `NodePort`. Here's an example that creates a `Service` with subtype `NodePort`:*

```
kubectl create service nodeport <myservicename>
```

*In the preceding example, the `create service nodeport` command is called a subcommand of the `create service` command.*

*You can use the `-h` flag to find the arguments and flags supported by a subcommand:*

```
kubectl create service nodeport -h
```

## **How to update objects**

*The `kubectl` command supports verb-driven commands for some common update operations. These commands are named to enable users unfamiliar with Kubernetes objects to perform updates without knowing the specific fields that must be set:*

- `scale`: Horizontally scale a controller to add or remove Pods by updating the replica count of the controller.
- `annotate`: Add or remove an annotation from an object.
- `label`: Add or remove a label from an object.

*The `kubectl` command also supports update commands driven by an aspect of the object. Setting this aspect may set different fields for different object types:*

- `set <field>`: Set an aspect of an object.

**Note:** In Kubernetes version 1.5, not every verb-driven command has an associated aspect-driven command.

*The `kubectl` tool supports these additional ways to update a live object directly, however they require a better understanding of the Kubernetes object schema.*

- `edit`: Directly edit the raw configuration of a live object by opening its configuration in an editor.

- **patch**: Directly modify specific fields of a live object by using a patch string. For more details on patch strings, see the patch section in [API Conventions](#).

## How to delete objects

You can use the `delete` command to delete an object from a cluster:

- `delete <type>/<name>`

**Note:** You can use `kubectl delete` for both imperative commands and imperative object configuration. The difference is in the arguments passed to the command. To use `kubectl delete` as an imperative command, pass the object to be deleted as an argument. Here's an example that passes a Deployment object named `nginx`:

```
kubectl delete deployment/nginx
```

## How to view an object

There are several commands for printing information about an object:

- **get**: Prints basic information about matching objects. Use `get -h` to see a list of options.
- **describe**: Prints aggregated detailed information about matching objects.
- **logs**: Prints the `stdout` and `stderr` for a container running in a Pod.

## Using set commands to modify objects before creation

There are some object fields that don't have a flag you can use in a `create` command. In some of those cases, you can use a combination of `set` and `create` to specify a value for the field before object creation. This is done by piping the output of the `create` command to the `set` command, and then back to the `create` command. Here's an example:

```
kubectl create service clusterip my-svc --clusterip="None" -o yaml --dry-run=client | kubectl set selector --local -f - 'environment=qa' -o yaml | kubectl create -f -
```

1. The `kubectl create service -o yaml --dry-run=client` command creates the configuration for the Service, but prints it to stdout as YAML instead of sending it to the Kubernetes API server.
2. The `kubectl set selector --local -f - -o yaml` command reads the configuration from stdin, and writes the updated configuration to stdout as YAML.
3. The `kubectl create -f -` command creates the object using the configuration provided via stdin.

## **Using `--edit` to modify objects before creation**

You can use `kubectl create --edit` to make arbitrary changes to an object before it is created. Here's an example:

```
kubectl create service clusterip my-svc --clusterip="None" -o yaml --dry-run=client > /tmp/srv.yaml  
kubectl create --edit -f /tmp/srv.yaml
```

1. The `kubectl create service` command creates the configuration for the Service and saves it to `/tmp/srv.yaml`.
2. The `kubectl create --edit` command opens the configuration file for editing before it creates the object.

## **What's next**

- [Managing Kubernetes Objects Using Object Configuration \(Imperative\)](#)
- [Managing Kubernetes Objects Using Object Configuration \(Declarative\)](#)
- [Kubectl Command Reference](#)
- [Kubernetes API 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](#).

Last modified September 24, 2020 at 8:05 AM PST: [Update the run command to refer to pod creation \(9fd549b12\)](#)  
[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Trade-offs](#)
- [How to create objects](#)
- [How to update objects](#)
- [How to delete objects](#)
- [How to view an object](#)
- [Using set commands to modify objects before creation](#)
- [Using --edit to modify objects before creation](#)
- [What's next](#)

# **Imperative Management of Kubernetes Objects Using Configuration Files**

Kubernetes objects can be created, updated, and deleted by using the `kubectl` command-line tool along with an object configuration file written in YAML or JSON. This document explains how to define and manage objects using configuration files.

## **Before you begin**

Install [kubectl](#).

You need to have a Kubernetes cluster, and the `kubectl` command-line tool must be configured to communicate with your cluster. If you do not already have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:

- [Katacoda](#)
- [Play with Kubernetes](#)

To check the version, enter `kubectl version`.

## **Trade-offs**

The `kubectl` tool supports three kinds of object management:

- *Imperative commands*
- *Imperative object configuration*
- *Declarative object configuration*

See [Kubernetes Object Management](#) for a discussion of the advantages and disadvantage of each kind of object management.

## How to create objects

You can use `kubectl create -f` to create an object from a configuration file. Refer to the [kubernetes API reference](#) for details.

- `kubectl create -f <filename/url>`

## How to update objects

**Warning:** Updating objects with the `replace` command drops all parts of the spec not specified in the configuration file. This should not be used with objects whose specs are partially managed by the cluster, such as Services of type `LoadBalancer`, where the `externalIPs` field is managed independently from the configuration file. Independently managed fields must be copied to the configuration file to prevent `replace` from dropping them.

You can use `kubectl replace -f` to update a live object according to a configuration file.

- `kubectl replace -f <filename/url>`

## How to delete objects

You can use `kubectl delete -f` to delete an object that is described in a configuration file.

- `kubectl delete -f <filename/url>`

### Note:

If configuration file has specified the `generateName` field in the `metadata` section instead of the `name` field, you cannot delete the object using `kubectl delete -f <filename/url>`. You will have to use other flags for deleting the object. For example:

```
kubectl delete <type> <name>
kubectl delete <type> -l <label>
```

## **How to view an object**

You can use `kubectl get -f` to view information about an object that is described in a configuration file.

- `kubectl get -f <filename/url> -o yaml`

The `-o yaml` flag specifies that the full object configuration is printed. Use `kubectl get -h` to see a list of options.

## **Limitations**

The `create`, `replace`, and `delete` commands work well when each object's configuration is fully defined and recorded in its configuration file. However when a live object is updated, and the updates are not merged into its configuration file, the updates will be lost the next time a `replace` is executed. This can happen if a controller, such as a `HorizontalPodAutoscaler`, makes updates directly to a live object. Here's an example:

1. You create an object from a configuration file.
2. Another source updates the object by changing some field.
3. You replace the object from the configuration file. Changes made by the other source in step 2 are lost.

If you need to support multiple writers to the same object, you can use `kubectl apply` to manage the object.

## **Creating and editing an object from a URL without saving the configuration**

Suppose you have the URL of an object configuration file. You can use `kubectl create --edit` to make changes to the configuration before the object is created. This is particularly useful for tutorials and tasks that point to a configuration file that could be modified by the reader.

```
kubectl create -f <url> --edit
```

# **Migrating from imperative commands to imperative object configuration**

*Migrating from imperative commands to imperative object configuration involves several manual steps.*

1. Export the live object to a local object configuration file:

```
kubectl get <kind>/<name> -o yaml > <kind>_<name>.yaml
```

2. Manually remove the status field from the object configuration file.

3. For subsequent object management, use `replace` exclusively.

```
kubectl replace -f <kind>_<name>.yaml
```

## **Defining controller selectors and PodTemplate labels**

**Warning:** Updating selectors on controllers is strongly discouraged.

The recommended approach is to define a single, immutable PodTemplate label used only by the controller selector with no other semantic meaning.

Example label:

```
selector:  
  matchLabels:  
    controller-selector: "apps/v1/deployment/nginx"  
template:  
  metadata:  
    labels:  
      controller-selector: "apps/v1/deployment/nginx"
```

## **What's next**

- [Managing Kubernetes Objects Using Imperative Commands](#)
- [Managing Kubernetes Objects Using Object Configuration \(Declarative\)](#)
- [Kubectl Command Reference](#)
- [Kubernetes API 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](#).

Last modified September 08, 2020 at 3:14 PM PST: [Improve object management page by adding a hint \(7ff239625\)](#)

[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Trade-offs](#)
- [How to create objects](#)
- [How to update objects](#)
- [How to delete objects](#)
- [How to view an object](#)
- [Limitations](#)
- [Creating and editing an object from a URL without saving the configuration](#)
- [Migrating from imperative commands to imperative object configuration](#)
- [Defining controller selectors and PodTemplate labels](#)
- [What's next](#)

# Update API Objects in Place Using kubectl patch

Use `kubectl patch` to update Kubernetes API objects in place. Do a strategic merge patch or a JSON merge patch.

This task shows how to use `kubectl patch` to update an API object in place. The exercises in this task demonstrate a strategic merge patch and a JSON merge patch.

## Before you begin

You need to have a Kubernetes cluster, and the `kubectl` command-line tool must be configured to communicate with your cluster. If you do not already have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:

- [Katacoda](#)
- [Play with Kubernetes](#)

To check the version, enter `kubectl version`.

## Use a strategic merge patch to update a Deployment

Here's the configuration file for a Deployment that has two replicas. Each replica is a Pod that has one container:

[application/deployment-patch.yaml](#)  


```
apiVersion: apps/v1 # for versions before 1.9.0 use apps/v1beta2
kind: Deployment
metadata:
  name: patch-demo
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: patch-demo-ctr
          image: nginx
      tolerations:
        - effect: NoSchedule
          key: dedicated
          value: test-team
```

Create the Deployment:

```
kubectl apply -f https://k8s.io/examples/application/deployment-patch.yaml
```

View the Pods associated with your Deployment:

```
kubectl get pods
```

The output shows that the Deployment has two Pods. The 1/1 indicates that each Pod has one container:

NAME	READY	STATUS	RESTARTS	AGE
patch-demo-28633765-670qr	1/1	Running	0	23s
patch-demo-28633765-j5qs3	1/1	Running	0	23s

Make a note of the names of the running Pods. Later, you will see that these Pods get terminated and replaced by new ones.

At this point, each Pod has one Container that runs the nginx image. Now suppose you want each Pod to have two containers: one that runs nginx and one that runs redis.

Create a file named `patch-file.yaml` that has this content:

```
spec:
  template:
    spec:
      containers:
        - name: patch-demo-ctr-2
          image: redis
```

Patch your Deployment:

- [Bash](#)
- [PowerShell](#)

```
kubectl patch deployment patch-demo --patch "$(cat patch-file.yaml)"
```

```
kubectl patch deployment patch-demo --patch $(Get-Content patch-file.yaml -Raw)
```

View the patched Deployment:

```
kubectl get deployment patch-demo --output yaml
```

The output shows that the PodSpec in the Deployment has two Containers:

```
containers:
- image: redis
  imagePullPolicy: Always
  name: patch-demo-ctr-2
  ...
```

```
- image: nginx
  imagePullPolicy: Always
  name: patch-demo-ctr
  ...

```

*View the Pods associated with your patched Deployment:*

```
kubectl get pods
```

*The output shows that the running Pods have different names from the Pods that were running previously. The Deployment terminated the old Pods and created two new Pods that comply with the updated Deployment spec. The 2 /2 indicates that each Pod has two Containers:*

NAME	READY	STATUS	RESTARTS	AGE
patch-demo-1081991389-2wrn5	2/2	Running	0	1m
patch-demo-1081991389-jmg7b	2/2	Running	0	1m

*Take a closer look at one of the patch-demo Pods:*

```
kubectl get pod <your-pod-name> --output yaml
```

*The output shows that the Pod has two Containers: one running nginx and one running redis:*

```
containers:
- image: redis
  ...
- image: nginx
  ...

```

## **Notes on the strategic merge patch**

*The patch you did in the preceding exercise is called a strategic merge patch. Notice that the patch did not replace the containers list. Instead it added a new Container to the list. In other words, the list in the patch was merged with the existing list. This is not always what happens when you use a strategic merge patch on a list. In some cases, the list is replaced, not merged.*

*With a strategic merge patch, a list is either replaced or merged depending on its patch strategy. The patch strategy is specified by the value of the patc*

*hStrategy* key in a field tag in the Kubernetes source code. For example, the *Containers* field of *PodSpec* struct has a *patchStrategy* of *merge*:

```
type PodSpec struct {  
    ...  
    Containers []Container `json:"containers"  
    patchStrategy:"merge" patchMergeKey:"name" ...`
```

You can also see the patch strategy in the [OpenAPI spec](#):

```
"io.k8s.api.core.v1.PodSpec": {  
    ...  
    "containers": {  
        "description": "List of containers belonging to the  
pod. ...  
    },  
    "x-kubernetes-patch-merge-key": "name",  
    "x-kubernetes-patch-strategy": "merge"  
},
```

And you can see the patch strategy in the [Kubernetes API documentation](#).

Create a file named *patch-file-tolerations.yaml* that has this content:

```
spec:  
  template:  
    spec:  
      tolerations:  
      - effect: NoSchedule  
        key: disktype  
        value: ssd
```

Patch your Deployment:

```
kubectl patch deployment patch-demo --patch "$(cat patch-file-  
tolerations.yaml)"
```

View the patched Deployment:

```
kubectl get deployment patch-demo --output yaml
```

The output shows that the *PodSpec* in the Deployment has only one Tolerance:

```
tolerations:
  - effect: NoSchedule
    key: disktype
    value: ssd
```

Notice that the `tolerations` list in the `PodSpec` was replaced, not merged. This is because the `Tolerations` field of `PodSpec` does not have a `patchStrategy` key in its field tag. So the strategic merge patch uses the default patch strategy, which is `replace`.

```
type PodSpec struct {
    ...
    Tolerations []Toleration `json:"tolerations,omitempty"`
    protobuf:"bytes,22,opt,name=tolerations"`}
```

## Use a JSON merge patch to update a Deployment

A strategic merge patch is different from a [JSON merge patch](#). With a JSON merge patch, if you want to update a list, you have to specify the entire new list. And the new list completely replaces the existing list.

The `kubectl patch` command has a `type` parameter that you can set to one of these values:

Parameter value	Merge type
json	<a href="#">JSON Patch, RFC 6902</a>
merge	<a href="#">JSON Merge Patch, RFC 7386</a>
strategic	Strategic merge patch

For a comparison of JSON patch and JSON merge patch, see [JSON Patch and JSON Merge Patch](#).

The default value for the `type` parameter is `strategic`. So in the preceding exercise, you did a strategic merge patch.

Next, do a JSON merge patch on your same Deployment. Create a file named `patch-file-2.yaml` that has this content:

```
spec:
  template:
    spec:
      containers:
```

- ```
- name: patch-demo-ctr-3
  image: gcr.io/google-samples/node-hello:1.0
```

In your patch command, set type to merge:

```
kubectl patch deployment patch-demo --type merge --patch "$(cat patch-file-2.yaml)"
```

View the patched Deployment:

```
kubectl get deployment patch-demo --output yaml
```

The containers list that you specified in the patch has only one Container. The output shows that your list of one Container replaced the existing containers list.

```
spec:
  containers:
  - image: gcr.io/google-samples/node-hello:1.0
    ...
  name: patch-demo-ctr-3
```

List the running Pods:

```
kubectl get pods
```

In the output, you can see that the existing Pods were terminated, and new Pods were created. The 1/1 indicates that each new Pod is running only one Container.

| NAME                        | READY | STATUS  | RESTARTS | AGE |
|-----------------------------|-------|---------|----------|-----|
| patch-demo-1307768864-69308 | 1/1   | Running | 0        | 1m  |
| patch-demo-1307768864-c86dc | 1/1   | Running | 0        | 1m  |

## Use strategic merge patch to update a Deployment using the retainKeys strategy

Here's the configuration file for a Deployment that uses the RollingUpdate strategy:

[application/deployment-retainkeys.yaml](#)



```
apiVersion: apps/v1 # for versions before 1.9.0 use apps/v1beta2
kind: Deployment
metadata:
  name: retainkeys-demo
spec:
  selector:
    matchLabels:
      app: nginx
  strategy:
    rollingUpdate:
      maxSurge: 30%
  template:
    metadata:
      labels:
        app: nginx
  spec:
    containers:
      - name: retainkeys-demo-ctr
        image: nginx
```

Create the deployment:

```
kubectl apply -f https://k8s.io/examples/application/deployment-retainkeys.yaml
```

At this point, the deployment is created and is using the RollingUpdate strategy.

Create a file named `patch-file-no-retainkeys.yaml` that has this content:

```
spec:
  strategy:
    type: Recreate
```

Patch your Deployment:

- [Bash](#)
- [PowerShell](#)

```
kubectl patch deployment retainkeys-demo --patch "$(cat patch-file-no-retainkeys.yaml)"
```

```
kubectl patch deployment retainkeys-demo --patch $(Get-Content patch-file-no-retainkeys.yaml -Raw)
```

In the output, you can see that it is not possible to set type as Recreate when a value is defined for spec.strategy.rollingUpdate:

```
The Deployment "retainkeys-demo" is invalid:  
spec.strategy.rollingUpdate: Forbidden: may not be specified  
when strategy `type` is 'Recreate'
```

The way to remove the value for spec.strategy.rollingUpdate when updating the value for type is to use the retainKeys strategy for the strategic merge.

Create another file named patch-file-retainkeys.yaml that has this content:

```
spec:  
  strategy:  
    $retainKeys:  
      - type  
        type: Recreate
```

With this patch, we indicate that we want to retain only the type key of the strategy object. Thus, the rollingUpdate will be removed during the patch operation.

Patch your Deployment again with this new patch:

- [Bash](#)
- [PowerShell](#)

```
kubectl patch deployment retainkeys-demo --patch "$(cat patch-file-retainkeys.yaml)"
```

```
kubectl patch deployment retainkeys-demo --patch $(Get-Content patch-file-retainkeys.yaml -Raw)
```

Examine the content of the Deployment:

```
kubectl get deployment retainkeys-demo --output yaml
```

The output shows that the strategy object in the Deployment does not contain the `rollingUpdate` key anymore:

```
spec:  
  strategy:  
    type: Recreate  
  template:
```

## Notes on the strategic merge patch using the `retainKeys` strategy

The patch you did in the preceding exercise is called a strategic merge patch with `retainKeys` strategy. This method introduces a new directive `$retainKeys` that has the following strategies:

- It contains a list of strings.
- All fields needing to be preserved must be present in the `$retainKeys` list.
- The fields that are present will be merged with live object.
- All of the missing fields will be cleared when patching.
- All fields in the `$retainKeys` list must be a superset or the same as the fields present in the patch.

The `retainKeys` strategy does not work for all objects. It only works when the value of the `patchStrategy` key in a field tag in the Kubernetes source code contains `retainKeys`. For example, the `Strategy` field of the `DeploymentSpec` struct has a `patchStrategy` of `retainKeys`:

```
type DeploymentSpec struct {  
  ...  
  // +patchStrategy=retainKeys  
  Strategy DeploymentStrategy `json:"strategy,omitempty"  
  patchStrategy:"retainKeys" ...
```

You can also see the `retainKeys` strategy in the [OpenApi spec](#):

```
"io.k8s.api.apps.v1.DeploymentSpec": {  
  ...  
  "strategy": {  
    "$ref": "#/definitions/  
    io.k8s.api.apps.v1.DeploymentStrategy",  
    "description": "The deployment strategy to use to replace  
existing pods with new ones.",
```

```
"x-kubernetes-patch-strategy": "retainKeys"  
},
```

And you can see the `retainKeys` strategy in the [Kubernetes API documentation](#).

## Alternate forms of the `kubectl patch` command

The `kubectl patch` command takes YAML or JSON. It can take the patch as a file or directly on the command line.

Create a file named `patch-file.json` that has this content:

```
{  
  "spec": {  
    "template": {  
      "spec": {  
        "containers": [  
          {  
            "name": "patch-demo-ctr-2",  
            "image": "redis"  
          }  
        ]  
      }  
    }  
  }  
}
```

The following commands are equivalent:

```
kubectl patch deployment patch-demo --patch "$(cat patch-file.yaml)"  
kubectl patch deployment patch-demo --patch 'spec:\n template:\n spec:\n   containers:\n     - name: patch-demo-ctr-2\n       image: redis'  
  
kubectl patch deployment patch-demo --patch "$(cat patch-file.json)"  
kubectl patch deployment patch-demo --patch '{"spec":\n  {"template": {"spec": {"containers": [{"name": "patch-demo-ctr-2", "image": "redis"}]}}}'
```

# **Summary**

*In this exercise, you used `kubectl patch` to change the live configuration of a Deployment object. You did not change the configuration file that you originally used to create the Deployment object. Other commands for updating API objects include [kubectl annotate](#), [kubectl edit](#), [kubectl replace](#), [kubectl scale](#), and [kubectl apply](#).*

## **What's next**

- [Kubernetes Object Management](#)
- [Managing Kubernetes Objects Using Imperative Commands](#)
- [Imperative Management of Kubernetes Objects Using Configuration Files](#)
- [Declarative Management of Kubernetes Objects Using Configuration Files](#)

## **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](#).*

*Last modified August 07, 2020 at 4:46 PM PST: [Tune links in tasks section \(1/2\) \(b8541d212\)](#)*

[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Use a strategic merge patch to update a Deployment](#)
  - [Notes on the strategic merge patch](#)
- [Use a JSON merge patch to update a Deployment](#)
- [Use strategic merge patch to update a Deployment using the retainKeys strategy](#)
  - [Notes on the strategic merge patch using the retainKeys strategy](#)
- [Alternate forms of the kubectl patch command](#)
- [Summary](#)
- [What's next](#)

# **Managing Secrets**

*Managing confidential settings data using Secrets.*

---

### [Managing Secret using kubectl](#)

*Creating Secret objects using kubectl command line.*

### [Managing Secret using Configuration File](#)

*Creating Secret objects using resource configuration file.*

### [Managing Secret using Kustomize](#)

*Creating Secret objects using kustomization.yaml file.*

# **Managing Secret using kubectl**

*Creating Secret objects using kubectl command line.*

## **Before you begin**

*You need to have a Kubernetes cluster, and the kubectl command-line tool must be configured to communicate with your cluster. If you do not already have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:*

- [Katacoda](#)
- [Play with Kubernetes](#)

## **Create a Secret**

*A Secret can contain user credentials required by Pods to access a database. For example, a database connection string consists of a username and password. You can store the username in a file ./username.txt and the password in a file ./password.txt on your local machine.*

```
echo -n 'admin' > ./username.txt
echo -n '1f2d1e2e67df' > ./password.txt
```

*The -n flag in the above two commands ensures that the generated files will not contain an extra newline character at the end of the text. This is important because when kubectl reads a file and encode the content into base64 string, the extra newline character gets encoded too.*

*The kubectl create secret command packages these files into a Secret and creates the object on the API server.*

```
kubectl create secret generic db-user-pass \
--from-file=./username.txt \
--from-file=./password.txt
```

The output is similar to:

```
secret/db-user-pass created
```

Default key name is the filename. You may optionally set the key name using `--from-file=[key=]source`. For example:

```
kubectl create secret generic db-user-pass \
--from-file=username=./username.txt \
--from-file=password=./password.txt
```

You do not need to escape special characters in passwords from files (`--from-file`).

You can also provide Secret data using the `--from-literal=<key>=<value>` tag. This tag can be specified more than once to provide multiple key-value pairs. Note that special characters such as \$, \, \*, =, and ! will be interpreted by your [shell](#) and require escaping. In most shells, the easiest way to escape the password is to surround it with single quotes (''). For example, if your actual password is `S!B\*d$zDsb=`, you should execute the command this way:

```
kubectl create secret generic dev-db-secret \
--from-literal=username=devuser \
--from-literal=password='S!B\*d$zDsb='
```

## Verify the Secret

You can check that the secret was created:

```
kubectl get secrets
```

The output is similar to:

| NAME         | TYPE   |
|--------------|--------|
| DATA AGE     |        |
| db-user-pass | Opaque |
| 2 51s        |        |

You can view a description of the Secret:

```
kubectl describe secrets/db-user-pass
```

The output is similar to:

|              |              |
|--------------|--------------|
| Name:        | db-user-pass |
| Namespace:   | default      |
| Labels:      | <none>       |
| Annotations: | <none>       |
| Type:        | Opaque       |
| Data         |              |

```
=====
password:    12 bytes
username:    5 bytes
```

The commands `kubectl get` and `kubectl describe` avoid showing the contents of a `Secret` by default. This is to protect the `Secret` from being exposed accidentally to an onlooker, or from being stored in a terminal log.

## Decoding the Secret

To view the contents of the `Secret` we just created, you can run the following command:

```
kubectl get secret db-user-pass -o jsonpath='{.data}'
```

The output is similar to:

```
{"password.txt": "MwYyZDFlMmU2N2Rm", "username.txt": "YWRtaW4="}
```

Now you can decode the `password.txt` data:

```
echo 'MwYyZDFlMmU2N2Rm' | base64 --decode
```

The output is similar to:

```
1f2d1e2e67df
```

## Clean Up

To delete the `Secret` you have just created:

```
kubectl delete secret db-user-pass
```

## What's next

- Read more about the [Secret concept](#)
- Learn how to [manage Secret using config file](#)
- Learn how to [manage Secret using kustomize](#)

## 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](#).

Last modified December 02, 2020 at 2:25 PM PST: [Corrected the field names in the secret \(43d071e8e\)](#)

- [Before you begin](#)
- [Create a Secret](#)
- [Verify the Secret](#)
- [Decoding the Secret](#)
- [Clean Up](#)
- [What's next](#)

# Managing Secret using Configuration File

*Creating Secret objects using resource configuration file.*

## Before you begin

*You need to have a Kubernetes cluster, and the kubectl command-line tool must be configured to communicate with your cluster. If you do not already have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:*

- [Katacoda](#)
- [Play with Kubernetes](#)

## Create the Config file

*You can create a Secret in a file first, in JSON or YAML format, and then create that object. The [Secret](#) resource contains two maps: `data` and `stringData`. The `data` field is used to store arbitrary data, encoded using base64. The `stringData` field is provided for convenience, and it allows you to provide Secret data as unencoded strings. The keys of `data` and `stringData` must consist of alphanumeric characters, `-`, `_` or `.`*

*For example, to store two strings in a Secret using the `data` field, convert the strings to base64 as follows:*

```
echo -n 'admin' | base64
```

*The output is similar to:*

```
YWRtaW4=
```

```
echo -n '1f2d1e2e67df' | base64
```

*The output is similar to:*

```
MWYyZDFlMmU2N2Rm
```

*Write a Secret config file that looks like this:*

```
apiVersion: v1
kind: Secret
metadata:
  name: mysecret
type: Opaque
data:
  username: YwRtaW4=
  password: MWWyZDFlMmU2N2Rm
```

Note that the name of a Secret object must be a valid [DNS subdomain name](#).

**Note:** The serialized JSON and YAML values of Secret data are encoded as base64 strings. Newlines are not valid within these strings and must be omitted. When using the `base64` utility on Darwin/macOS, users should avoid using the `-b` option to split long lines. Conversely, Linux users should add the option `-w 0` to `base64` commands or the pipeline `base64 | tr -d '\n'` if the `-w` option is not available.

For certain scenarios, you may wish to use the `stringData` field instead. This field allows you to put a non-base64 encoded string directly into the Secret, and the string will be encoded for you when the Secret is created or updated.

A practical example of this might be where you are deploying an application that uses a Secret to store a configuration file, and you want to populate parts of that configuration file during your deployment process.

For example, if your application uses the following configuration file:

```
apiUrl: "https://my.api.com/api/v1"
username: "<user>"
password: "<password>"
```

You could store this in a Secret using the following definition:

```
apiVersion: v1
kind: Secret
metadata:
  name: mysecret
type: Opaque
stringData:
  config.yaml: |
    apiUrl: "https://my.api.com/api/v1"
    username: <user>
    password: <password>
```

## Create the Secret object

Now create the Secret using [`kubectl apply`](#):

```
kubectl apply -f ./secret.yaml
```

The output is similar to:

```
secret/mysecret created
```

## Check the Secret

The `stringData` field is a write-only convenience field. It is never output when retrieving Secrets. For example, if you run the following command:

```
kubectl get secret mysecret -o yaml
```

The output is similar to:

```
apiVersion: v1
kind: Secret
metadata:
  creationTimestamp: 2018-11-15T20:40:59Z
  name: mysecret
  namespace: default
  resourceVersion: "7225"
  uid: c280ad2e-e916-11e8-98f2-025000000001
type: Opaque
data:
  config.yaml: YXBpVXJs0iAiaHR0cHM6Ly9teS5hcGkuY29tL2FwaS92MSIKdXNlcm5hbWU6IHt7dXNlcm5hbWV9fQpwYXNzd29yZDoge3twYXNzd29yZH19
```

The commands `kubectl get` and `kubectl describe` avoid showing the contents of a `Secret` by default. This is to protect the `Secret` from being exposed accidentally to an onlooker, or from being stored in a terminal log. To check the actual content of the encoded data, please refer to [decoding secret](#).

If a field, such as `username`, is specified in both `data` and `stringData`, the value from `stringData` is used. For example, the following `Secret` definition:

```
apiVersion: v1
kind: Secret
metadata:
  name: mysecret
type: Opaque
data:
  username: YWRtaW4=
stringData:
  username: administrator
```

Results in the following `Secret`:

```
apiVersion: v1
kind: Secret
metadata:
  creationTimestamp: 2018-11-15T20:46:46Z
  name: mysecret
```

```
namespace: default
resourceVersion: "7579"
uid: 91460ecb-e917-11e8-98f2-025000000001
type: Opaque
data:
  username: YWRtaW5pc3RyYXRvcg==
```

Where `YWRtaW5pc3RyYXRvcg==` decodes to `administrator`.

## Clean Up

To delete the Secret you have just created:

```
kubectl delete secret db-user-pass
```

## What's next

- Read more about the [Secret concept](#)
- Learn how to [manage Secret with the kubectl command](#)
- Learn how to [manage Secret using kustomize](#)

## 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](#).

Last modified September 19, 2020 at 12:18 PM PST: [Fixed typo \(cf14321b7\)](#)  
[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Create the Config file](#)
- [Create the Secret object](#)
- [Check the Secret](#)
- [Clean Up](#)
- [What's next](#)

# Managing Secret using Kustomize

Creating Secret objects using `kustomization.yaml` file.

Since Kubernetes v1.14, `kubectl` supports [managing objects using Kustomize](#). Kustomize provides resource Generators to create Secrets and ConfigMaps. The Kustomize generators should be specified in a `kustomizat`

*ion.yaml* file inside a directory. After generating the Secret, you can create the Secret on the API server with `kubectl apply`.

## Before you begin

You need to have a Kubernetes cluster, and the `kubectl` command-line tool must be configured to communicate with your cluster. If you do not already have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:

- [Katacoda](#)
- [Play with Kubernetes](#)

## Create the Kustomization file

You can generate a Secret by defining a `secretGenerator` in a `kustomization.yaml` file that references other existing files. For example, the following kustomization file references the `./username.txt` and the `./password.txt` files:

```
secretGenerator:  
- name: db-user-pass  
  files:  
    - username.txt  
    - password.txt
```

You can also define the `secretGenerator` in the `kustomization.yaml` file by providing some literals. For example, the following `kustomization.yaml` file contains two literals for `username` and `password` respectively:

```
secretGenerator:  
- name: db-user-pass  
  literals:  
    - username=admin  
    - password=1f2d1e2e67df
```

Note that in both cases, you don't need to base64 encode the values.

## Create the Secret

Apply the directory containing the `kustomization.yaml` to create the Secret.

```
kubectl apply -k .
```

The output is similar to:

```
secret/db-user-pass-96mffmfh4k created
```

*Note that when a Secret is generated, the Secret name is created by hashing the Secret data and appending the hash value to the name. This ensures that a new Secret is generated each time the data is modified.*

## **Check the Secret created**

*You can check that the secret was created:*

```
kubectl get secrets
```

*The output is similar to:*

| NAME                    | DATA | AGE |
|-------------------------|------|-----|
| db-user-pass-96mffmfh4k |      |     |
| Opaque                  | 2    | 51s |

*You can view a description of the secret:*

```
kubectl describe secrets/db-user-pass-96mffmfh4k
```

*The output is similar to:*

```
Name: db-user-pass
Namespace: default
Labels: <none>
Annotations: <none>

Type: Opaque

Data
=====
password.txt: 12 bytes
username.txt: 5 bytes
```

*The commands kubectl get and kubectl describe avoid showing the contents of a Secret by default. This is to protect the Secret from being exposed accidentally to an onlooker, or from being stored in a terminal log. To check the actual content of the encoded data, please refer to [decoding secret](#).*

## **Clean Up**

*To delete the Secret you have just created:*

```
kubectl delete secret db-user-pass-96mffmfh4k
```

## **What's next**

- Read more about the [Secret concept](#)
- Learn how to [manage Secret with the kubectl command](#)

- Learn how to [manage Secret using config file](#)

## 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](#).

Last modified September 10, 2020 at 2:27 PM PST: [Separating tasks from Secret concept \(995b067fd\)](#)

[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Create the Kustomization file](#)
- [Create the Secret](#)
- [Check the Secret created](#)
- [Clean Up](#)
- [What's next](#)

# Inject Data Into Applications

Specify configuration and other data for the Pods that run your workload.

---

[Define a Command and Arguments for a Container](#)

[Define Dependent Environment Variables](#)

[Define Environment Variables for a Container](#)

[Expose Pod Information to Containers Through Environment Variables](#)

[Expose Pod Information to Containers Through Files](#)

[Distribute Credentials Securely Using Secrets](#)

[Inject Information into Pods Using a PodPreset](#)

## Define a Command and Arguments for a Container

This page shows how to define commands and arguments when you run a container in a [Pod](#).

# **Before you begin**

You need to have a Kubernetes cluster, and the `kubectl` command-line tool must be configured to communicate with your cluster. If you do not already have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:

- [Katacoda](#)
- [Play with Kubernetes](#)

To check the version, enter `kubectl version`.

## **Define a command and arguments when you create a Pod**

When you create a Pod, you can define a command and arguments for the containers that run in the Pod. To define a command, include the `command` field in the configuration file. To define arguments for the command, include the `args` field in the configuration file. The command and arguments that you define cannot be changed after the Pod is created.

The command and arguments that you define in the configuration file override the default command and arguments provided by the container image. If you define args, but do not define a command, the default command is used with your new arguments.

**Note:** The `command` field corresponds to `entrypoint` in some container runtimes. Refer to the [Notes](#) below.

In this exercise, you create a Pod that runs one container. The configuration file for the Pod defines a command and two arguments:

[`pods/commands.yaml`](#)



```
apiVersion: v1
kind: Pod
metadata:
  name: command-demo
  labels:
    purpose: demonstrate-command
spec:
  containers:
    - name: command-demo-container
      image: debian
      command: ["printenv"]
```

```
args: ["HOSTNAME", "KUBERNETES_PORT"]
restartPolicy: OnFailure
```

1. Create a Pod based on the YAML configuration file:

```
kubectl apply -f https://k8s.io/examples/pods/commands.yaml
```

2. List the running Pods:

```
kubectl get pods
```

The output shows that the container that ran in the command-demo Pod has completed.

3. To see the output of the command that ran in the container, view the logs from the Pod:

```
kubectl logs command-demo
```

The output shows the values of the HOSTNAME and KUBERNETES\_PORT environment variables:

```
command-demo
tcp://10.3.240.1:443
```

## Use environment variables to define arguments

In the preceding example, you defined the arguments directly by providing strings. As an alternative to providing strings directly, you can define arguments by using environment variables:

```
env:
- name: MESSAGE
  value: "hello world"
  command: ["/bin/echo"]
  args: ["$(MESSAGE)"]
```

This means you can define an argument for a Pod using any of the techniques available for defining environment variables, including [ConfigMaps](#) and [Secrets](#).

**Note:** The environment variable appears in parentheses, "\$ (VAR)". This is required for the variable to be expanded in the command or args field.

# Run a command in a shell

In some cases, you need your command to run in a shell. For example, your command might consist of several commands piped together, or it might be a shell script. To run your command in a shell, wrap it like this:

```
command: ["/bin/sh"]
args: ["-c", "while true; do echo hello; sleep 10;done"]
```

## Notes

This table summarizes the field names used by Docker and Kubernetes.

| Description                         | Docker field name | Kubernetes field name |
|-------------------------------------|-------------------|-----------------------|
| The command run by the container    | Entrypoint        | command               |
| The arguments passed to the command | Cmd               | args                  |

When you override the default `Entrypoint` and `Cmd`, these rules apply:

- If you do not supply `command` or `args` for a Container, the defaults defined in the Docker image are used.
- If you supply a `command` but no `args` for a Container, only the supplied `command` is used. The default `EntryPoint` and the default `Cmd` defined in the Docker image are ignored.
- If you supply only `args` for a Container, the default `Entrypoint` defined in the Docker image is run with the `args` that you supplied.
- If you supply a `command` and `args`, the default `Entrypoint` and the default `Cmd` defined in the Docker image are ignored. Your `command` is run with your `args`.

Here are some examples:

| Image Entrypoint | Image Cmd | Container command | Container args | Command run    |
|------------------|-----------|-------------------|----------------|----------------|
| [/ep-1]          | [foo bar] | <not set>         | <not set>      | [ep-1 foo bar] |
| [/ep-1]          | [foo bar] | [/ep-2]           | <not set>      | [ep-2]         |
| [/ep-1]          | [foo bar] | <not set>         | [zoo boo]      | [ep-1 zoo boo] |

| <b>Image Entrypoint</b> | <b>Image Cmd</b>       | <b>Container command</b> | <b>Container args</b>  | <b>Command run</b>              |
|-------------------------|------------------------|--------------------------|------------------------|---------------------------------|
| <code>[/ep-1]</code>    | <code>[foo bar]</code> | <code>[/ep-2]</code>     | <code>[zoo boo]</code> | <code>[ep-2 zoo<br/>boo]</code> |

## What's next

- Learn more about [configuring pods and containers](#).
- Learn more about [running commands in a container](#).
- See [Container](#).

## 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](#).

Last modified May 30, 2020 at 3:10 PM PST: [add en pages \(ecc27bbbe\)](#)  
[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Define a command and arguments when you create a Pod](#)
- [Use environment variables to define arguments](#)
- [Run a command in a shell](#)
- [Notes](#)
- [What's next](#)

# Define Dependent Environment Variables

This page shows how to define dependent environment variables for a container in a Kubernetes Pod.

## Before you begin

You need to have a Kubernetes cluster, and the `kubectl` command-line tool must be configured to communicate with your cluster. If you do not already have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:

- [Katacoda](#)

- [Play with Kubernetes](#)

## Define an environment dependent variable for a container

When you create a Pod, you can set dependent environment variables for the containers that run in the Pod. To set dependent environment variables, you can use `$(VAR_NAME)` in the value of env in the configuration file.

In this exercise, you create a Pod that runs one container. The configuration file for the Pod defines an dependent environment variable with common usage defined. Here is the configuration manifest for the Pod:

[pods/inject/dependent-envvars.yaml](#)

```
apiVersion: v1
kind: Pod
metadata:
  name: dependent-envvars-demo
spec:
  containers:
    - name: dependent-envvars-demo
      args:
        - while true; do echo -en '\n'; printf UNCHANGED_REFERENCE=$UNCHANGED_REFERENCE'\n'; printf SERVICE_ADDRESS=$SERVICE_ADDRESS'\n';printf ESCAPED_REFERENCE=$ESCAPED_REFERENCE'\n'; sleep 30; done;
      command:
        - sh
        - -c
      image: busybox
      env:
        - name: SERVICE_PORT
          value: "80"
        - name: SERVICE_IP
          value: "172.17.0.1"
        - name: UNCHANGED_REFERENCE
          value: "$(PROTOCOL)://$(SERVICE_IP):$(SERVICE_PORT)"
        - name: PROTOCOL
          value: "https"
        - name: SERVICE_ADDRESS
          value: "$(PROTOCOL)://$(SERVICE_IP):$(SERVICE_PORT)"
        - name: ESCAPED_REFERENCE
          value: "$$(PROTOCOL)://$(SERVICE_IP):$(SERVICE_PORT)"
```

1. Create a Pod based on that manifest:

```
kubectl apply -f https://k8s.io/examples/pods/inject/
dependent-envvars.yaml
```

```
pod/dependent-envars-demo created
```

2. List the running Pods:

```
kubectl get pods dependent-envars-demo
```

| NAME                  | READY | STATUS  | RESTARTS | AGE |
|-----------------------|-------|---------|----------|-----|
| dependent-envars-demo | 1/1   | Running | 0        | 9s  |

3. Check the logs for the container running in your Pod:

```
kubectl logs pod/dependent-envars-demo
```

```
UNCHANGED_REFERENCE=$(PROTOCOL)://172.17.0.1:80
SERVICE_ADDRESS=https://172.17.0.1:80
ESCAPED_REFERENCE=$(PROTOCOL)://172.17.0.1:80
```

As shown above, you have defined the correct dependency reference of `SERVICE_ADDRESS`, bad dependency reference of `UNCHANGED_REFERENCE` and skip dependent references of `ESCAPED_REFERENCE`.

When an environment variable is already defined when being referenced, the reference can be correctly resolved, such as in the `SERVICE_ADDRESS` case.

When the environment variable is undefined or only includes some variables, the undefined environment variable is treated as a normal string, such as `UNCHANGED_REFERENCE`. Note that incorrectly parsed environment variables, in general, will not block the container from starting.

The `$(VAR_NAME)` syntax can be escaped with a double \$, ie: `$(VAR_NAME)`. Escaped references are never expanded, regardless of whether the referenced variable is defined or not. This can be seen from the `ESCAPED_REFERENCE` case above.

## What's next

- Learn more about [environment variables](#).
- See [EnvVarSource](#).

## 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](#).

Last modified July 01, 2020 at 2:48 PM PST: [use printf to replace printenv \(4490fe014\)](#)

[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Define an environment dependent variable for a container](#)
- [What's next](#)

# Define Environment Variables for a Container

This page shows how to define environment variables for a container in a Kubernetes Pod.

## Before you begin

You need to have a Kubernetes cluster, and the `kubectl` command-line tool must be configured to communicate with your cluster. If you do not already have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:

- [Katacoda](#)
- [Play with Kubernetes](#)

## Define an environment variable for a container

When you create a Pod, you can set environment variables for the containers that run in the Pod. To set environment variables, include the `env` or `envFrom` field in the configuration file.

In this exercise, you create a Pod that runs one container. The configuration file for the Pod defines an environment variable with name `DEMO_GREETING` and value "Hello from the environment". Here is the configuration manifest for the Pod:

[pods/inject/envvars.yaml](#)



```
apiVersion: v1
kind: Pod
metadata:
  name: envvar-demo
  labels:
    purpose: demonstrate-envvars
spec:
  containers:
    - name: envvar-demo-container
```

```
image: gcr.io/google-samples/node-hello:1.0
env:
- name: DEMO_GREETING
  value: "Hello from the environment"
- name: DEMO_FAREWELL
  value: "Such a sweet sorrow"
```

1. Create a Pod based on that manifest:

```
kubectl apply -f https://k8s.io/examples/pods/inject/
envvars.yaml
```

2. List the running Pods:

```
kubectl get pods -l purpose=demonstrate-envvars
```

The output is similar to:

| NAME       | READY | STATUS  | RESTARTS | AGE |
|------------|-------|---------|----------|-----|
| envar-demo | 1/1   | Running | 0        | 9s  |

3. List the Pod's container environment variables:

```
kubectl exec envar-demo -- printenv
```

The output is similar to this:

```
NODE_VERSION=4.4.2
EXAMPLE_SERVICE_PORT_8080_TCP_ADDR=10.3.245.237
HOSTNAME=envar-demo
...
DEMO_GREETING=Hello from the environment
DEMO_FAREWELL=Such a sweet sorrow
```

**Note:** The environment variables set using the `env` or `envFrom` field override any environment variables specified in the container image.

**Note:** The environment variables can reference each other, and cycles are possible, pay attention to the order before using

## Using environment variables inside of your config

Environment variables that you define in a Pod's configuration can be used elsewhere in the configuration, for example in commands and arguments that you set for the Pod's containers. In the example configuration below, the `GREETING`, `HONORIFIC`, and `NAME` environment variables are set to `Warm greetings to`, `The Most Honorable`, and `Kubernetes`, respectively. Those environment variables are then used in the CLI arguments passed to the `env-print-demo` container.

```
apiVersion: v1
kind: Pod
metadata:
  name: print-greeting
spec:
  containers:
    - name: env-print-demo
      image: bash
      env:
        - name: GREETING
          value: "Warm greetings to"
        - name: HONORIFIC
          value: "The Most Honorable"
        - name: NAME
          value: "Kubernetes"
      command: ["echo"]
      args: ["$(GREETING) $(HONORIFIC) $(NAME)"]
```

Upon creation, the command `echo Warm greetings to The Most Honorable Kubernetes` is run on the container.

## What's next

- Learn more about [environment variables](#).
- Learn about [using secrets as environment variables](#).
- See [EnvVarSource](#).

## 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](#).

Last modified October 23, 2020 at 10:51 AM PST: [Update content/en/docs/tasks/inject-data-application/define-environment-variable-container.md \(8cc80bf46\)](#)

[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Define an environment variable for a container](#)
- [Using environment variables inside of your config](#)
- [What's next](#)

# **Expose Pod Information to Containers Through Environment Variables**

*This page shows how a Pod can use environment variables to expose information about itself to Containers running in the Pod. Environment variables can expose Pod fields and Container fields.*

## **Before you begin**

*You need to have a Kubernetes cluster, and the `kubectl` command-line tool must be configured to communicate with your cluster. If you do not already have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:*

- [Katacoda](#)
- [Play with Kubernetes](#)

*To check the version, enter `kubectl version`.*

## **The Downward API**

*There are two ways to expose Pod and Container fields to a running Container:*

- *Environment variables*
- [Volume Files](#)

*Together, these two ways of exposing Pod and Container fields are called the Downward API.*

## **Use Pod fields as values for environment variables**

*In this exercise, you create a Pod that has one Container. Here is the configuration file for the Pod:*

[`pods/inject/dapi-envars-pod.yaml`](#)  


```
apiVersion: v1
kind: Pod
```

```

metadata:
  name: dapi-envars-fieldref
spec:
  containers:
    - name: test-container
      image: k8s.gcr.io/busybox
      command: [ "sh", "-c"]
      args:
        - while true; do
            echo -en '\n';
            printenv MY_NODE_NAME MY_POD_NAME MY_POD_NAMESPACE;
            printenv MY_POD_IP MY_POD_SERVICE_ACCOUNT;
            sleep 10;
        done;
  env:
    - name: MY_NODE_NAME
      valueFrom:
        fieldRef:
          fieldPath: spec.nodeName
    - name: MY_POD_NAME
      valueFrom:
        fieldRef:
          fieldPath: metadata.name
    - name: MY_POD_NAMESPACE
      valueFrom:
        fieldRef:
          fieldPath: metadata.namespace
    - name: MY_POD_IP
      valueFrom:
        fieldRef:
          fieldPath: status.podIP
    - name: MY_POD_SERVICE_ACCOUNT
      valueFrom:
        fieldRef:
          fieldPath: spec.serviceAccountName
  restartPolicy: Never

```

In the configuration file, you can see five environment variables. The `env` field is an array of [EnvVars](#). The first element in the array specifies that the `MY_NODE_NAME` environment variable gets its value from the Pod's `spec.nodeName` field. Similarly, the other environment variables get their names from Pod fields.

**Note:** The fields in this example are Pod fields. They are not fields of the Container in the Pod.

Create the Pod:

```
kubectl apply -f https://k8s.io/examples/pods/inject/dapi-envvars-pod.yaml
```

Verify that the Container in the Pod is running:

```
kubectl get pods
```

View the Container's logs:

```
kubectl logs dapi-envvars-fieldref
```

The output shows the values of selected environment variables:

```
minikube
dapi-envvars-fieldref
default
172.17.0.4
default
```

To see why these values are in the log, look at the `command` and `args` fields in the configuration file. When the Container starts, it writes the values of five environment variables to `stdout`. It repeats this every ten seconds.

Next, get a shell into the Container that is running in your Pod:

```
kubectl exec -it dapi-envvars-fieldref -- sh
```

In your shell, view the environment variables:

```
/# printenv
```

The output shows that certain environment variables have been assigned the values of Pod fields:

```
MY_POD_SERVICE_ACCOUNT=default
...
MY_POD_NAMESPACE=default
MY_POD_IP=172.17.0.4
...
MY_NODE_NAME=minikube
...
MY_POD_NAME=dapi-envvars-fieldref
```

## Use Container fields as values for environment variables

In the preceding exercise, you used Pod fields as the values for environment variables. In this next exercise, you use Container fields as the values for environment variables. Here is the configuration file for a Pod that has one container:

[pods/inject/dapi-envars-container.yaml](#)  


```
apiVersion: v1
kind: Pod
metadata:
  name: dapi-envars-resourcefieldref
spec:
  containers:
    - name: test-container
      image: k8s.gcr.io/busybox:1.24
      command: [ "sh", "-c" ]
      args:
        - while true; do
            echo -en '\n';
            printenv MY_CPU_REQUEST MY_CPU_LIMIT;
            printenv MY_MEM_REQUEST MY_MEM_LIMIT;
            sleep 10;
        done;
      resources:
        requests:
          memory: "32Mi"
          cpu: "125m"
        limits:
          memory: "64Mi"
          cpu: "250m"
    env:
      - name: MY_CPU_REQUEST
        valueFrom:
          resourceFieldRef:
            containerName: test-container
            resource: requests.cpu
      - name: MY_CPU_LIMIT
        valueFrom:
          resourceFieldRef:
            containerName: test-container
            resource: limits.cpu
      - name: MY_MEM_REQUEST
        valueFrom:
          resourceFieldRef:
            containerName: test-container
            resource: requests.memory
```

```
- name: MY_MEM_LIMIT
  valueFrom:
    resourceFieldRef:
      containerName: test-container
      resource: limits.memory
restartPolicy: Never
```

In the configuration file, you can see four environment variables. The `env` field is an array of [EnvVars](#). The first element in the array specifies that the `MY_CPU_REQUEST` environment variable gets its value from the `requests.cpu` field of a Container named `test-container`. Similarly, the other environment variables get their values from Container fields.

Create the Pod:

```
kubectl apply -f https://k8s.io/examples/pods/inject/dapi-envvars-
container.yaml
```

Verify that the Container in the Pod is running:

```
kubectl get pods
```

View the Container's logs:

```
kubectl logs dapi-envvars-resourcefieldref
```

The output shows the values of selected environment variables:

```
1
1
33554432
67108864
```

## What's next

- [Defining Environment Variables for a Container](#)
- [PodSpec](#)
- [Container](#)
- [EnvVar](#)
- [EnvVarSource](#)
- [ObjectFieldSelector](#)
- [ResourceFieldSelector](#)

# 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](#).

Last modified May 30, 2020 at 3:10 PM PST: [add en pages \(ecc27bbbe\)](#)  
[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [The Downward API](#)
- [Use Pod fields as values for environment variables](#)
- [Use Container fields as values for environment variables](#)
- [What's next](#)

# Expose Pod Information to Containers Through Files

This page shows how a Pod can use a `DownwardAPIVolumeFile` to expose information about itself to Containers running in the Pod. A `DownwardAPIVolumeFile` can expose Pod fields and Container fields.

## Before you begin

You need to have a Kubernetes cluster, and the `kubectl` command-line tool must be configured to communicate with your cluster. If you do not already have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:

- [Katacoda](#)
- [Play with Kubernetes](#)

To check the version, enter `kubectl version`.

## The Downward API

There are two ways to expose Pod and Container fields to a running Container:

- [Environment variables](#)
- [Volume Files](#)

Together, these two ways of exposing Pod and Container fields are called the Downward API.

## Store Pod fields

In this exercise, you create a Pod that has one Container. Here is the configuration file for the Pod:

[pods/inject/dapi-volume.yaml](#)



```
apiVersion: v1
kind: Pod
metadata:
  name: kubernetes-downwardapi-volume-example
  labels:
    zone: us-est-coast
    cluster: test-cluster1
    rack: rack-22
  annotations:
    build: two
    builder: john-doe
spec:
  containers:
    - name: client-container
      image: k8s.gcr.io/busybox
      command: ["sh", "-c"]
      args:
        - while true; do
            if [[ -e /etc/podinfo/labels ]]; then
              echo -en '\n\n'; cat /etc/podinfo/labels; fi;
            if [[ -e /etc/podinfo/annotations ]]; then
              echo -en '\n\n'; cat /etc/podinfo/annotations; fi;
            sleep 5;
          done;
      volumeMounts:
        - name: podinfo
          mountPath: /etc/podinfo
  volumes:
    - name: podinfo
  downwardAPI:
    items:
      - path: "labels"
        fieldRef:
          fieldPath: metadata.labels
      - path: "annotations"
        fieldRef:
          fieldPath: metadata.annotations
```

In the configuration file, you can see that the Pod has a `downwardAPI` Volume, and the Container mounts the Volume at `/etc/podinfo`.

Look at the `items` array under `downwardAPI`. Each element of the array is a [DownwardAPIVolumeFile](#). The first element specifies that the value of the Pod's `metadata.labels` field should be stored in a file named `labels`. The second element specifies that the value of the Pod's `annotations` field should be stored in a file named `annotations`.

**Note:** The fields in this example are Pod fields. They are not fields of the Container in the Pod.

Create the Pod:

```
kubectl apply -f https://k8s.io/examples/pods/inject/dapi-volume.yaml
```

Verify that the Container in the Pod is running:

```
kubectl get pods
```

View the Container's logs:

```
kubectl logs kubernetes-downwardapi-volume-example
```

The output shows the contents of the `labels` file and the `annotations` file:

```
cluster="test-cluster1"
rack="rack-22"
zone="us-est-coast"

build="two"
builder="john-doe"
```

Get a shell into the Container that is running in your Pod:

```
kubectl exec -it kubernetes-downwardapi-volume-example -- sh
```

In your shell, view the `labels` file:

```
#!/ cat /etc/podinfo/labels
```

The output shows that all of the Pod's labels have been written to the `labels` file:

```
cluster="test-cluster1"
rack="rack-22"
zone="us-est-coast"
```

Similarly, view the `annotations` file:

```
/# cat /etc/podinfo/annotations
```

View the files in the `/etc/podinfo` directory:

```
/# ls -laR /etc/podinfo
```

In the output, you can see that the `labels` and `annotations` files are in a temporary subdirectory: in this example, `..2982_06_02_21_47_53.299460680`. In the `/etc/podinfo` directory, `..data` is a symbolic link to the temporary subdirectory. Also in the `/etc/podinfo` directory, `labels` and `annotations` are symbolic links.

```
drwxr-xr-x  ... Feb  6 21:47 ..2982_06_02_21_47_53.299460680
lrwxrwxrwx  ... Feb  6 21:47 ..data -> ..
2982_06_02_21_47_53.299460680
lrwxrwxrwx  ... Feb  6 21:47 annotations -> ..data/annotations
lrwxrwxrwx  ... Feb  6 21:47 labels -> ..data/labels

/etc/..2982_06_02_21_47_53.299460680:
total 8
-rw-r--r--  ... Feb  6 21:47 annotations
-rw-r--r--  ... Feb  6 21:47 labels
```

Using symbolic links enables dynamic atomic refresh of the metadata; updates are written to a new temporary directory, and the `..data` symlink is updated atomically using [rename\(2\)](#).

**Note:** A container using Downward API as a [subPath](#) volume mount will not receive Downward API updates.

Exit the shell:

```
/# exit
```

## Store Container fields

The preceding exercise, you stored Pod fields in a `DownwardAPIVolumeFile`. In this next exercise, you store Container fields. Here is the configuration file for a Pod that has one Container:

[pods/inject/dapi-volume-resources.yaml](#)  


```
apiVersion: v1
kind: Pod
metadata:
  name: kubernetes-downwardapi-volume-example-2
spec:
  containers:
    - name: client-container
      image: k8s.gcr.io/busybox:1.24
      command: ["sh", "-c"]
      args:
        - while true; do
            echo -en '\n';
            if [[ -e /etc/podinfo/cpu_limit ]]; then
              echo -en '\n'; cat /etc/podinfo/cpu_limit; fi;
            if [[ -e /etc/podinfo/cpu_request ]]; then
              echo -en '\n'; cat /etc/podinfo/cpu_request; fi;
            if [[ -e /etc/podinfo/mem_limit ]]; then
              echo -en '\n'; cat /etc/podinfo/mem_limit; fi;
            if [[ -e /etc/podinfo/mem_request ]]; then
              echo -en '\n'; cat /etc/podinfo/mem_request; fi;
            sleep 5;
          done;
      resources:
        requests:
          memory: "32Mi"
          cpu: "125m"
        limits:
          memory: "64Mi"
          cpu: "250m"
      volumeMounts:
        - name: podinfo
          mountPath: /etc/podinfo
    volumes:
      - name: podinfo
    downwardAPI:
      items:
        - path: "cpu_limit"
          resourceFieldRef:
            containerName: client-container
            resource: limits.cpu
            divisor: 1m
```

```
- path: "cpu_request"
  resourceFieldRef:
    containerName: client-container
    resource: requests.cpu
    divisor: 1m
- path: "mem_limit"
  resourceFieldRef:
    containerName: client-container
    resource: limits.memory
    divisor: 1Mi
- path: "mem_request"
  resourceFieldRef:
    containerName: client-container
    resource: requests.memory
    divisor: 1Mi
```

In the configuration file, you can see that the Pod has a `downwardAPI` Volume, and the Container mounts the Volume at `/etc/podinfo`.

Look at the `items` array under `downwardAPI`. Each element of the array is a `DownwardAPIVolumeFile`.

The first element specifies that in the Container named `client-container`, the value of the `limits.cpu` field in the format specified by `1m` should be stored in a file named `cpu_limit`. The `divisor` field is optional and has the default value of `1` which means cores for cpu and bytes for memory.

Create the Pod:

```
kubectl apply -f https://k8s.io/examples/pods/inject/dapi-volume-resources.yaml
```

Get a shell into the Container that is running in your Pod:

```
kubectl exec -it kubernetes-downwardapi-volume-example-2 -- sh
```

In your shell, view the `cpu_limit` file:

```
#!/ cat /etc/podinfo/cpu_limit
```

You can use similar commands to view the `cpu_request`, `mem_limit` and `mem_request` files.

# **Capabilities of the Downward API**

The following information is available to containers through environment variables and downwardAPI volumes:

- Information available via `fieldRef`:
  - `metadata.name` - the pod's name
  - `metadata.namespace` - the pod's namespace
  - `metadata.uid` - the pod's UID
  - `metadata.labels['<KEY>']` - the value of the pod's label `<KEY>` (for example, `metadata.labels['mylabel']`)
  - `metadata.annotations['<KEY>']` - the value of the pod's annotation `<KEY>` (for example, `metadata.annotations['myannotation']`)
- Information available via `resourceFieldRef`:
  - A Container's CPU limit
  - A Container's CPU request
  - A Container's memory limit
  - A Container's memory request
  - A Container's hugepages limit (providing that the [DownwardAPI HugePages feature gate](#) is enabled)
  - A Container's hugepages request (providing that the [DownwardAPI HugePages feature gate](#) is enabled)
  - A Container's ephemeral-storage limit
  - A Container's ephemeral-storage request

In addition, the following information is available through downwardAPI volume `fieldRef`:

- `metadata.labels` - all of the pod's labels, formatted as `label-key="escaped-label-value"` with one label per line
- `metadata.annotations` - all of the pod's annotations, formatted as `annotation-key="escaped-annotation-value"` with one annotation per line

The following information is available through environment variables:

- `status.podIP` - the pod's IP address
- `spec.serviceAccountName` - the pod's service account name, available since v1.4.0-alpha.3
- `spec.nodeName` - the node's name, available since v1.4.0-alpha.3
- `status.hostIP` - the node's IP, available since v1.7.0-alpha.1

**Note:** If CPU and memory limits are not specified for a Container, the Downward API defaults to the node allocatable value for CPU and memory.

# **Project keys to specific paths and file permissions**

You can project keys to specific paths and specific permissions on a per-file basis. For more information, see [Secrets](#).

## **Motivation for the Downward API**

*It is sometimes useful for a Container to have information about itself, without being overly coupled to Kubernetes. The Downward API allows containers to consume information about themselves or the cluster without using the Kubernetes client or API server.*

*An example is an existing application that assumes a particular well-known environment variable holds a unique identifier. One possibility is to wrap the application, but that is tedious and error prone, and it violates the goal of low coupling. A better option would be to use the Pod's name as an identifier, and inject the Pod's name into the well-known environment variable.*

## **What's next**

- [PodSpec](#)
- [Volume](#)
- [DownwardAPIVolumeSource](#)
- [DownwardAPIVolumeFile](#)
- [ResourceFieldSelector](#)

## **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](#).

Last modified November 05, 2020 at 1:52 PM PST: [Add documentation for downward API hugepages \(6b4ab1790\)](#)

- [Before you begin](#)
- [The Downward API](#)
- [Store Pod fields](#)
- [Store Container fields](#)
- [Capabilities of the Downward API](#)
- [Project keys to specific paths and file permissions](#)
- [Motivation for the Downward API](#)
- [What's next](#)

# Distribute Credentials Securely Using Secrets

This page shows how to securely inject sensitive data, such as passwords and encryption keys, into Pods.

## Before you begin

You need to have a Kubernetes cluster, and the `kubectl` command-line tool must be configured to communicate with your cluster. If you do not already have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:

- [Katacoda](#)
- [Play with Kubernetes](#)

## Convert your secret data to a base-64 representation

Suppose you want to have two pieces of secret data: a username `my-app` and a password `39528$vdg7Jb`. First, use a base64 encoding tool to convert your username and password to a base64 representation. Here's an example using the commonly available `base64` program:

```
echo -n 'my-app' | base64  
echo -n '39528$vdg7Jb' | base64
```

The output shows that the base-64 representation of your username is `bXktYXBw`, and the base-64 representation of your password is `Mzk1MjgkdmRnN0pi`.

**Caution:** Use a local tool trusted by your OS to decrease the security risks of external tools.

## Create a Secret

Here is a configuration file you can use to create a Secret that holds your username and password:



```
apiVersion: v1
kind: Secret
metadata:
  name: test-secret
data:
  username: bXktYXBw
  password: Mzk1MjgkdmRnN0pi
```

### 1. Create the Secret

```
kubectl apply -f https://k8s.io/examples/pods/inject/
secret.yaml
```

### 2. View information about the Secret:

```
kubectl get secret test-secret
```

*Output:*

| NAME        | TYPE   | DATA | AGE |
|-------------|--------|------|-----|
| test-secret | Opaque | 2    | 1m  |

### 3. View more detailed information about the Secret:

```
kubectl describe secret test-secret
```

*Output:*

```
Name:      test-secret
Namespace: default
Labels:    <none>
Annotations: <none>

Type:     Opaque

Data
=====
password:  13 bytes
username:  7 bytes
```

## **Create a Secret directly with kubectl**

If you want to skip the Base64 encoding step, you can create the same Secret using the `kubectl create secret` command. For example:

```
kubectl create secret generic test-secret --from-literal='username=my-app' --from-literal='password=39528$vdg7Jb'
```

This is more convenient. The detailed approach shown earlier runs through each step explicitly to demonstrate what is happening.

# Create a Pod that has access to the secret data through a Volume

Here is a configuration file you can use to create a Pod:

[pods/inject/secret-pod.yaml](#)



```
apiVersion: v1
kind: Pod
metadata:
  name: secret-test-pod
spec:
  containers:
    - name: test-container
      image: nginx
      volumeMounts:
        # name must match the volume name below
        - name: secret-volume
          mountPath: /etc/secret-volume
      # The secret data is exposed to Containers in the Pod through
      # a Volume.
    volumes:
      - name: secret-volume
        secret:
          secretName: test-secret
```

1. Create the Pod:

```
kubectl apply -f https://k8s.io/examples/pods/inject/secret-
pod.yaml
```

2. Verify that your Pod is running:

```
kubectl get pod secret-test-pod
```

Output:

| NAME            | READY | STATUS  | RESTARTS | AGE |
|-----------------|-------|---------|----------|-----|
| secret-test-pod | 1/1   | Running | 0        | 42m |

3. Get a shell into the Container that is running in your Pod:

```
kubectl exec -i -t secret-test-pod -- /bin/bash
```

4. The secret data is exposed to the Container through a Volume mounted under /etc/secret-volume.

In your shell, list the files in the /etc/secret-volume directory:

```
# Run this in the shell inside the container
ls /etc/secret-volume
```

The output shows two files, one for each piece of secret data:

```
password username
```

5. In your shell, display the contents of the `username` and `password` files:

```
# Run this in the shell inside the container
echo "$( cat /etc/secret-volume/username )"
echo "$( cat /etc/secret-volume/password )"
```

The output is your `username` and `password`:

```
my-app
39528$vdg7Jb
```

## Define container environment variables using Secret data

### Define a container environment variable with data from a single Secret

- Define an environment variable as a key-value pair in a Secret:

```
kubectl create secret generic backend-user --from-literal=backend-username='backend-admin'
```

- Assign the `backend-username` value defined in the Secret to the `SECRET_USERNAME` environment variable in the Pod specification.

[pods/inject/pod-single-secret-env-variable.yaml](#)



```
apiVersion: v1
kind: Pod
metadata:
  name: env-single-secret
spec:
  containers:
    - name: envvars-test-container
      image: nginx
      env:
        - name: SECRET_USERNAME
          valueFrom:
            secretKeyRef:
              name: backend-user
              key: backend-username
```

- Create the Pod:

```
kubectl create -f https://k8s.io/examples/pods/inject/pod-single-secret-env-variable.yaml
```

- In your shell, display the content of `SECRET_USERNAME` container environment variable

```
kubectl exec -i -t env-single-secret -- /bin/sh -c 'echo $SECRET_USERNAME'
```

The output is

```
backend-admin
```

## Define container environment variables with data from multiple Secrets

- As with the previous example, create the Secrets first.

```
kubectl create secret generic backend-user --from-literal=backend-username='backend-admin'  
kubectl create secret generic db-user --from-literal=db-username='db-admin'
```

- Define the environment variables in the Pod specification.

[pods/inject/pod-multiple-secret-env-variable.yaml](#)



```
apiVersion: v1  
kind: Pod  
metadata:  
  name: envvars-multiple-secrets  
spec:  
  containers:  
    - name: envars-test-container  
      image: nginx  
      env:  
        - name: BACKEND_USERNAME  
          valueFrom:  
            secretKeyRef:  
              name: backend-user  
              key: backend-username  
        - name: DB_USERNAME  
          valueFrom:  
            secretKeyRef:  
              name: db-user  
              key: db-username
```

- Create the Pod:

```
kubectl create -f https://k8s.io/examples/pods/inject/pod-multiple-secret-env-variable.yaml
```

- In your shell, display the container environment variables

```
kubectl exec -i -t envvars-multiple-secrets -- /bin/sh -c 'env | grep _USERNAME'
```

The output is

```
DB_USERNAME=db-admin
BACKEND_USERNAME=backend-admin
```

## Configure all key-value pairs in a Secret as container environment variables

**Note:** This functionality is available in Kubernetes v1.6 and later.

- Create a Secret containing multiple key-value pairs

```
kubectl create secret generic test-secret --from-literal=user
name='my-app' --from-literal=password='39528$vdg7Jb'
```

- Use `envFrom` to define all of the Secret's data as container environment variables. The key from the Secret becomes the environment variable name in the Pod.

[pods/inject/pod-secret-envFrom.yaml](#)  


```
apiVersion: v1
kind: Pod
metadata:
  name: envfrom-secret
spec:
  containers:
    - name: envvars-test-container
      image: nginx
      envFrom:
        - secretRef:
            name: test-secret
```

- Create the Pod:

```
kubectl create -f https://k8s.io/examples/pods/inject/pod-
secret-envFrom.yaml
```

- In your shell, display `username` and `password` container environment variables

```
kubectl exec -i -t envfrom-secret -- /bin/sh -c 'echo
"username: $username\npassword: $password\n"'
```

The output is

```
username: my-app
password: 39528$vdg7Jb
```

## References

- [Secret](#)
- [Volume](#)
- [Pod](#)

## What's next

- Learn more about [Secrets](#).
- Learn about [Volumes](#).

## 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](#).

Last modified August 05, 2020 at 3:17 AM PST: [Replace special quote characters with normal ones. \(c6a96128c\)](#)

[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
  - [Convert your secret data to a base-64 representation](#)
- [Create a Secret](#)
  - [Create a Secret directly with kubectl](#)
- [Create a Pod that has access to the secret data through a Volume](#)
- [Define container environment variables using Secret data](#)
  - [Define a container environment variable with data from a single Secret](#)
  - [Define container environment variables with data from multiple Secrets](#)
- [Configure all key-value pairs in a Secret as container environment variables](#)
  - [References](#)
- [What's next](#)

# Inject Information into Pods Using a PodPreset

**FEATURE STATE:** Kubernetes v1.6 [alpha]

This page shows how to use PodPreset objects to inject information like [Secrets](#), volume mounts, and [environment variables](#) into Pods at creation time.

# Before you begin

You need to have a Kubernetes cluster, and the `kubectl` command-line tool must be configured to communicate with your cluster. If you do not already have a cluster, you can create one using [Minikube](#). Make sure that you have [enabled PodPreset](#) in your cluster.

## Use Pod presets to inject environment variables and volumes

In this step, you create a preset that has a volume mount and one environment variable. Here is the manifest for the PodPreset:

[podpreset/preset.yaml](#)



```
apiVersion: settings.k8s.io/v1alpha1
kind: PodPreset
metadata:
  name: allow-database
spec:
  selector:
    matchLabels:
      role: frontend
  env:
    - name: DB_PORT
      value: "6379"
  volumeMounts:
    - mountPath: /cache
      name: cache-volume
  volumes:
    - name: cache-volume
      emptyDir: {}
```

The name of a PodPreset object must be a valid [DNS subdomain name](#).

In the manifest, you can see that the preset has an environment variable definition called `DB_PORT` and a volume mount definition called `cache-volume` which is mounted under `/cache`. The `selector` specifies that the preset will act upon any Pod that is labeled `role:frontend`.

Create the PodPreset:

```
kubectl apply -f https://k8s.io/examples/podpreset/preset.yaml
```

Verify that the PodPreset has been created:

```
kubectl get podpreset
```

| NAME           | CREATED AT           |
|----------------|----------------------|
| allow-database | 2020-01-24T08:54:29Z |

This manifest defines a Pod labelled `role: frontend` (matching the PodPreset's selector):

[podpreset/pod.yaml](#)



```
apiVersion: v1
kind: Pod
metadata:
  name: website
  labels:
    app: website
    role: frontend
spec:
  containers:
    - name: website
      image: nginx
      ports:
        - containerPort: 80
```

Create the Pod:

```
kubectl create -f https://k8s.io/examples/podpreset/pod.yaml
```

Verify that the Pod is running:

```
kubectl get pods
```

The output shows that the Pod is running:

| NAME    | READY | STATUS  | RESTARTS | AGE |
|---------|-------|---------|----------|-----|
| website | 1/1   | Running | 0        | 4m  |

View the Pod spec altered by the admission controller in order to see the effects of the preset having been applied:

```
kubectl get pod website -o yaml
```

[podpreset/merged.yaml](#)



```
apiVersion: v1
kind: Pod
metadata:
  name: website
  labels:
    app: website
    role: frontend
  annotations:
    podpreset.admission.kubernetes.io/podpreset-allow-database: "resource version"
```

```

spec:
  containers:
    - name: website
      image: nginx
      volumeMounts:
        - mountPath: /cache
          name: cache-volume
      ports:
        - containerPort: 80
      env:
        - name: DB_PORT
          value: "6379"
  volumes:
    - name: cache-volume
      emptyDir: {}

```

The `DB_PORT` environment variable, the `volumeMount` and the `podpreset.admission.kubernetes.io` annotation of the Pod verify that the preset has been applied.

## **Pod spec with ConfigMap example**

This is an example to show how a Pod spec is modified by a Pod preset that references a ConfigMap containing environment variables.

Here is the manifest containing the definition of the ConfigMap:

[podpreset/configmap.yaml](#)



```

apiVersion: v1
kind: ConfigMap
metadata:
  name: etcd-env-config
data:
  number_of_members: "1"
  initial_cluster_state: new
  initial_cluster_token: DUMMY_ETCD_INITIAL_CLUSTER_TOKEN
  discovery_token: DUMMY_ETCD_DISCOVERY_TOKEN
  discovery_url: http://etcd_discovery:2379
  etcdctl_peers: http://etcd:2379
  duplicate_key: FROM_CONFIG_MAP
  REPLACE_ME: "a value"

```

Create the ConfigMap:

```
kubectl create -f https://k8s.io/examples/podpreset/
configmap.yaml
```

Here is a PodPreset manifest referencing that ConfigMap:

[\*podpreset/allow-db.yaml\*](#)



```
apiVersion: settings.k8s.io/v1alpha1
kind: PodPreset
metadata:
  name: allow-database
spec:
  selector:
    matchLabels:
      role: frontend
  env:
    - name: DB_PORT
      value: "6379"
    - name: duplicate_key
      value: FROM_ENV
    - name: expansion
      value: $(REPLACE_ME)
  envFrom:
    - configMapRef:
        name: etcd-env-config
  volumeMounts:
    - mountPath: /cache
      name: cache-volume
  volumes:
    - name: cache-volume
      emptyDir: {}
```

Create the preset that references the ConfigMap:

```
kubectl create -f https://k8s.io/examples/podpreset/allow-db.yaml
```

The following manifest defines a Pod matching the PodPreset for this example:

[\*podpreset/pod.yaml\*](#)



```
apiVersion: v1
kind: Pod
metadata:
  name: website
  labels:
    app: website
    role: frontend
spec:
  containers:
    - name: website
      image: nginx
      ports:
        - containerPort: 80
```

Create the Pod:

```
kubectl create -f https://k8s.io/examples/podpreset/pod.yaml
```

View the Pod spec altered by the admission controller in order to see the effects of the preset having been applied:

```
kubectl get pod website -o yaml
```

[podpreset/allow-db-merged.yaml](#)



```
apiVersion: v1
kind: Pod
metadata:
  name: website
  labels:
    app: website
    role: frontend
  annotations:
    podpreset.admission.kubernetes.io/podpreset-allow-database: ""
resource version:
spec:
  containers:
    - name: website
      image: nginx
      volumeMounts:
        - mountPath: /cache
          name: cache-volume
      ports:
        - containerPort: 80
      env:
        - name: DB_PORT
          value: "6379"
        - name: duplicate_key
          value: FROM_ENV
        - name: expansion
          value: $(REPLACE_ME)
      envFrom:
        - configMapRef:
            name: etcd-env-config
  volumes:
    - name: cache-volume
      emptyDir: {}
```

The `DB_PORT` environment variable and the `podpreset.admission.kubernetes.io` annotation of the Pod verify that the preset has been applied.

## ReplicaSet with Pod spec example

This is an example to show that only Pod specs are modified by Pod presets. Other workload types like ReplicaSets or Deployments are unaffected.

Here is the manifest for the PodPreset for this example:

[podpreset/preset.yaml](#)



```
apiVersion: settings.k8s.io/v1alpha1
kind: PodPreset
metadata:
  name: allow-database
spec:
  selector:
    matchLabels:
      role: frontend
  env:
    - name: DB_PORT
      value: "6379"
  volumeMounts:
    - mountPath: /cache
      name: cache-volume
  volumes:
    - name: cache-volume
      emptyDir: {}
```

Create the preset:

```
kubectl apply -f https://k8s.io/examples/podpreset/preset.yaml
```

This manifest defines a ReplicaSet that manages three application Pods:

[podpreset/replicaset.yaml](#)



```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: frontend
spec:
  replicas: 3
  selector:
    matchLabels:
      role: frontend
    matchExpressions:
      - {key: role, operator: In, values: [frontend]}
  template:
    metadata:
      labels:
```

```

  app: guestbook
  role: frontend
spec:
  containers:
    - name: php-redis
      image: gcr.io/google_samples/gb-frontend:v3
      resources:
        requests:
          cpu: 100m
          memory: 100Mi
      env:
        - name: GET_HOSTS_FROM
          value: dns
      ports:
        - containerPort: 80

```

Create the ReplicaSet:

```
kubectl create -f https://k8s.io/examples/podpreset/
replicaset.yaml
```

Verify that the Pods created by the ReplicaSet are running:

```
kubectl get pods
```

The output shows that the Pods are running:

| NAME           | READY | STATUS  | RESTARTS | AGE   |
|----------------|-------|---------|----------|-------|
| frontend-2l94q | 1/1   | Running | 0        | 2m18s |
| frontend-6vdgn | 1/1   | Running | 0        | 2m18s |
| frontend-jzt4p | 1/1   | Running | 0        | 2m18s |

View the spec of the ReplicaSet:

```
kubectl get replicaset frontend -o yaml
```

### Note:

The ReplicaSet object's spec was not changed, nor does the ReplicaSet contain a podpreset.admission.kubernetes.io annotation. This is because a PodPreset only applies to Pod objects.

To see the effects of the preset having been applied, you need to look at individual Pods.

The command to view the specs of the affected Pods is:

```
kubectl get pod --selector=role=frontend -o yaml
```

[podpreset/replaset-merged.yaml](#)



```

apiVersion: v1
kind: Pod
metadata:
  name: frontend
  labels:
    app: guestbook
    role: frontend
  annotations:
    podpreset.admission.kubernetes.io/podpreset-allow-database: "resource version"
spec:
  containers:
    - name: php-redis
      image: gcr.io/google_samples/gb-frontend:v3
      resources:
        requests:
          cpu: 100m
          memory: 100Mi
      volumeMounts:
        - mountPath: /cache
          name: cache-volume
      env:
        - name: GET_HOSTS_FROM
          value: dns
        - name: DB_PORT
          value: "6379"
      ports:
        - containerPort: 80
    volumes:
      - name: cache-volume
        emptyDir: {}

```

Again the `podpreset.admission.kubernetes.io` annotation of the Pods verifies that the preset has been applied.

## **Multiple Pod presets example**

This is an example to show how a Pod spec is modified by multiple Pod presets.

Here is the manifest for the first PodPreset:

[podpreset/preset.yaml](#)



```

apiVersion: settings.k8s.io/v1alpha1
kind: PodPreset

```

```
metadata:  
  name: allow-database  
spec:  
  selector:  
    matchLabels:  
      role: frontend  
  env:  
    - name: DB_PORT  
      value: "6379"  
  volumeMounts:  
    - mountPath: /cache  
      name: cache-volume  
  volumes:  
    - name: cache-volume  
  emptyDir: {}
```

Create the first PodPreset for this example:

```
kubectl apply -f https://k8s.io/examples/podpreset/preset.yaml
```

Here is the manifest for the second PodPreset:

[podpreset/proxy.yaml](#)



```
apiVersion: settings.k8s.io/v1alpha1  
kind: PodPreset  
metadata:  
  name: proxy  
spec:  
  selector:  
    matchLabels:  
      role: frontend  
  volumeMounts:  
    - mountPath: /etc/proxy/configs  
      name: proxy-volume  
  volumes:  
    - name: proxy-volume  
  emptyDir: {}
```

Create the second preset:

```
kubectl apply -f https://k8s.io/examples/podpreset/proxy.yaml
```

Here's a manifest containing the definition of an applicable Pod (matched by two PodPresets):

[podpreset/pod.yaml](#)



```
apiVersion: v1  
kind: Pod
```

```

metadata:
  name: website
  labels:
    app: website
    role: frontend
spec:
  containers:
    - name: website
      image: nginx
      ports:
        - containerPort: 80

```

Create the Pod:

```
kubectl create -f https://k8s.io/examples/podpreset/pod.yaml
```

View the Pod spec altered by the admission controller in order to see the effects of both presets having been applied:

```
kubectl get pod website -o yaml
```

[podpreset/multi-merged.yaml](#)



```

apiVersion: v1
kind: Pod
metadata:
  name: website
  labels:
    app: website
    role: frontend
  annotations:
    podpreset.admission.kubernetes.io/podpreset-allow-database: ""
resource version:
    podpreset.admission.kubernetes.io/podpreset-proxy: "resource
version:
spec:
  containers:
    - name: website
      image: nginx
      volumeMounts:
        - mountPath: /cache
          name: cache-volume
        - mountPath: /etc/proxy/configs
          name: proxy-volume
      ports:
        - containerPort: 80
      env:
        - name: DB_PORT
          value: "6379"
    volumes:

```

- ```

- name: cache-volume
  emptyDir: {}
- name: proxy-volume
  emptyDir: {}

```

The `DB_PORT` environment variable, the `proxy-volume` `VolumeMount` and the two `podpreset.admission.kubernetes.io` annotations of the Pod verify that both presets have been applied.

## Conflict example

This is an example to show how a Pod spec is not modified by a Pod preset when there is a conflict. The conflict in this example consists of a `VolumeMount` in the PodPreset conflicting with a Pod that defines the same `mountPath`.

Here is the manifest for the PodPreset:

[podpreset/conflict-preset.yaml](#)  


```

apiVersion: settings.k8s.io/v1alpha1
kind: PodPreset
metadata:
  name: allow-database
spec:
  selector:
    matchLabels:
      role: frontend
  env:
    - name: DB_PORT
      value: "6379"
  volumeMounts:
    - mountPath: /cache
      name: other-volume
  volumes:
    - name: other-volume
      emptyDir: {}

```

Note the `mountPath` value of `/cache`.

Create the preset:

```
kubectl apply -f https://k8s.io/examples/podpreset/conflict-
preset.yaml
```

Here is the manifest for the Pod:

[podpreset/conflict-pod.yaml](#)  


```

apiVersion: v1
kind: Pod
metadata:
  name: website
  labels:
    app: website
    role: frontend
spec:
  containers:
    - name: website
      image: nginx
      volumeMounts:
        - mountPath: /cache
          name: cache-volume
      ports:
        - containerPort: 80
  volumes:
    - name: cache-volume
      emptyDir: {}

```

Note the volumeMount element with the same path as in the PodPreset.

Create the Pod:

```
kubectl create -f https://k8s.io/examples/podpreset/conflict-pod.yaml
```

View the Pod spec:

```
kubectl get pod website -o yaml
```

[podpreset/conflict-pod.yaml](#)



```

apiVersion: v1
kind: Pod
metadata:
  name: website
  labels:
    app: website
    role: frontend
spec:
  containers:
    - name: website
      image: nginx
      volumeMounts:
        - mountPath: /cache
          name: cache-volume
      ports:
        - containerPort: 80
  volumes:

```

```
- name: cache-volume  
emptyDir: {}
```

You can see there is no preset annotation (`podpreset.admission.kubernetes.io`). Seeing no annotation tells you that no preset has not been applied to the Pod.

However, the [PodPreset admission controller](#) logs a warning containing details of the conflict. You can view the warning using `kubectl`:

```
kubectl -n kube-system logs -l=component=kube-apiserver
```

The output should look similar to:

```
W1214 13:00:12.987884      1 admission.go:147] conflict  
occurred while applying podpresets: allow-database on pod: err:  
merging volume mounts for allow-database has a conflict on mount  
path /cache:  
v1.VolumeMount{Name:"other-volume", ReadOnly:false, MountPath:"/cache", SubPath:"", MountPropagation:(*v1.MountPropagationMode)(nil), SubPathExpr:""}  
does not match  
core.VolumeMount{Name:"cache-volume", ReadOnly:false, MountPath:"/cache", SubPath:"", MountPropagation:(*core.MountPropagationMode)(nil), SubPathExpr:""}  
in container
```

Note the conflict message on the path for the `VolumeMount`.

## Deleting a PodPreset

Once you don't need a PodPreset anymore, you can delete it with `kubectl`:

```
kubectl delete podpreset allow-database
```

The output shows that the PodPreset was deleted:

```
podpreset "allow-database" deleted
```

## 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](#).

Last modified October 22, 2020 at 3:01 PM PST: [Fix links in the tasks section \(740eb340d\)](#)

- [Before you begin](#)
- [Use Pod presets to inject environment variables and volumes](#)
- [Pod spec with ConfigMap example](#)
- [ReplicaSet with Pod spec example](#)
- [Multiple Pod presets example](#)
- [Conflict example](#)
- [Deleting a PodPreset](#)

# Run Applications

Run and manage both stateless and stateful applications.

---

[\*\*Run a Stateless Application Using a Deployment\*\*](#)

[\*\*Run a Single-Instance Stateful Application\*\*](#)

[\*\*Run a Replicated Stateful Application\*\*](#)

[\*\*Scale a StatefulSet\*\*](#)

[\*\*Delete a StatefulSet\*\*](#)

[\*\*Force Delete StatefulSet Pods\*\*](#)

[\*\*Horizontal Pod Autoscaler\*\*](#)

[\*\*Horizontal Pod Autoscaler Walkthrough\*\*](#)

[\*\*Specifying a Disruption Budget for your Application\*\*](#)

# Run a Stateless Application Using a Deployment

This page shows how to run an application using a Kubernetes Deployment object.

## Objectives

- Create an nginx deployment.
- Use kubectl to list information about the deployment.
- Update the deployment.

## Before you begin

You need to have a Kubernetes cluster, and the `kubectl` command-line tool must be configured to communicate with your cluster. If you do not already have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:

- [Katacoda](#)
- [Play with Kubernetes](#)

Your Kubernetes server must be at or later than version v1.9. To check the version, enter `kubectl version`.

## **Creating and exploring an nginx deployment**

You can run an application by creating a Kubernetes Deployment object, and you can describe a Deployment in a YAML file. For example, this YAML file describes a Deployment that runs the `nginx:1.14.2` Docker image:

[\*application/deployment.yaml\*](#)



```
apiVersion: apps/v1 # for versions before 1.9.0 use apps/v1beta2
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 2 # tells deployment to run 2 pods matching the template
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:1.14.2
          ports:
            - containerPort: 80
```

1. Create a Deployment based on the YAML file:

```
kubectl apply -f https://k8s.io/examples/application/deployment.yaml
```

2. Display information about the Deployment:

```
kubectl describe deployment nginx-deployment
```

The output is similar to this:

```
user@computer:~/website$ kubectl describe deployment nginx-deployment
Name:      nginx-deployment
Namespace:  default
CreationTimestamp: Tue, 30 Aug 2016 18:11:37 -0700
Labels:    app=nginx
Annotations: deployment.kubernetes.io/revision=1
Selector:  app=nginx
Replicas:  2 desired | 2 updated | 2 total | 2 available |
0 unavailable
StrategyType: RollingUpdate
MinReadySeconds: 0
RollingUpdateStrategy: 1 max unavailable, 1 max surge
Pod Template:
  Labels:    app=nginx
  Containers:
    nginx:
      Image:      nginx:1.14.2
      Port:       80/TCP
      Environment: <none>
      Mounts:     <none>
      Volumes:    <none>
  Conditions:
    Type        Status  Reason
    ----        ----   -----
    Available   True    MinimumReplicasAvailable
    Progressing True    NewReplicaSetAvailable
    OldReplicaSets: <none>
    NewReplicaSet:  nginx-deployment-1771418926 (2/2 replicas created)
    No events.
```

3. List the Pods created by the deployment:

```
kubectl get pods -l app=nginx
```

The output is similar to this:

NAME	READY	STATUS
RESTARTS	AGE	
nginx-deployment-1771418926-7o5ns	1/1	Running
0 16h		
nginx-deployment-1771418926-r18az	1/1	Running
0 16h		

4. Display information about a Pod:

```
kubectl describe pod <pod-name>
```

where `<pod-name>` is the name of one of your Pods.

## Updating the deployment

You can update the deployment by applying a new YAML file. This YAML file specifies that the deployment should be updated to use nginx 1.16.1.

[application/deployment-update.yaml](#)



```
apiVersion: apps/v1 # for versions before 1.9.0 use apps/v1beta2
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 2
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:1.16.1 # Update the version of nginx from
1.14.2 to 1.16.1
        ports:
          - containerPort: 80
```

1. Apply the new YAML file:

```
kubectl apply -f https://k8s.io/examples/application/
deployment-update.yaml
```

2. Watch the deployment create pods with new names and delete the old pods:

```
kubectl get pods -l app=nginx
```

## Scaling the application by increasing the replica count

You can increase the number of Pods in your Deployment by applying a new YAML file. This YAML file sets `replicas` to 4, which specifies that the Deployment should have four Pods:

[application/deployment-scale.yaml](#)



```
apiVersion: apps/v1 # for versions before 1.9.0 use apps/v1beta2
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 4 # Update the replicas from 2 to 4
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:1.14.2
          ports:
            - containerPort: 80
```

1. Apply the new YAML file:

```
kubectl apply -f https://k8s.io/examples/application/
deployment-scale.yaml
```

2. Verify that the Deployment has four Pods:

```
kubectl get pods -l app=nginx
```

The output is similar to this:

NAME	READY	STATUS
RESTARTS	AGE	
nginx-deployment-148880595-4zdqq	1/1	Running
0	25s	
nginx-deployment-148880595-6zgil	1/1	Running
0	25s	
nginx-deployment-148880595-fxcez	1/1	Running
0	2m	
nginx-deployment-148880595-rwovn	1/1	Running
0	2m	

## **Deleting a deployment**

Delete the deployment by name:

```
kubectl delete deployment nginx-deployment
```

## ReplicationControllers -- the Old Way

The preferred way to create a replicated application is to use a Deployment, which in turn uses a ReplicaSet. Before the Deployment and ReplicaSet were added to Kubernetes, replicated applications were configured using a [ReplicationController](#).

## What's next

- Learn more about [Deployment 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](#).

Last modified September 15, 2020 at 2:59 PM PST: [Update run-stateless-application-deployment.md \(42f23e75b\)](#)

[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Objectives](#)
- [Before you begin](#)
- [Creating and exploring an nginx deployment](#)
- [Updating the deployment](#)
- [Scaling the application by increasing the replica count](#)
- [Deleting a deployment](#)
- [ReplicationControllers -- the Old Way](#)
- [What's next](#)

## Run a Single-Instance Stateful Application

This page shows you how to run a single-instance stateful application in Kubernetes using a PersistentVolume and a Deployment. The application is MySQL.

# Objectives

- Create a PersistentVolume referencing a disk in your environment.
- Create a MySQL Deployment.
- Expose MySQL to other pods in the cluster at a known DNS name.

## Before you begin

- You need to have a Kubernetes cluster, and the `kubectl` command-line tool must be configured to communicate with your cluster. If you do not already have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:
  - [Katacoda](#)
  - [Play with Kubernetes](#)
- To check the version, enter `kubectl version`.
- You need to either have a dynamic PersistentVolume provisioner with a default [StorageClass](#), or [statically provision PersistentVolumes](#) yourself to satisfy the [PersistentVolumeClaims](#) used here.

## Deploy MySQL

You can run a stateful application by creating a Kubernetes Deployment and connecting it to an existing PersistentVolume using a PersistentVolumeClaim. For example, this YAML file describes a Deployment that runs MySQL and references the PersistentVolumeClaim. The file defines a volume mount for `/var/lib/mysql`, and then creates a PersistentVolumeClaim that looks for a 20G volume. This claim is satisfied by any existing volume that meets the requirements, or by a dynamic provisioner.

Note: The password is defined in the config yaml, and this is insecure. See [Kubernetes Secrets](#) for a secure solution.

[application/mysql/mysql-deployment.yaml](#)  


```
apiVersion: v1
kind: Service
metadata:
  name: mysql
spec:
  ports:
  - port: 3306
  selector:
    app: mysql
  clusterIP: None
---
apiVersion: apps/v1 # for versions before 1.9.0 use apps/v1beta2
```

```

kind: Deployment
metadata:
  name: mysql
spec:
  selector:
    matchLabels:
      app: mysql
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: mysql
    spec:
      containers:
        - image: mysql:5.6
        name: mysql
        env:
          # Use secret in real usage
          - name: MYSQL_ROOT_PASSWORD
            value: password
      ports:
        - containerPort: 3306
          name: mysql
      volumeMounts:
        - name: mysql-persistent-storage
          mountPath: /var/lib/mysql
      volumes:
        - name: mysql-persistent-storage
          persistentVolumeClaim:
            claimName: mysql-pv-claim

```

[application/mysql/mysql-pv.yaml](#)



```

apiVersion: v1
kind: PersistentVolume
metadata:
  name: mysql-pv-volume
  labels:
    type: local
spec:
  storageClassName: manual
  capacity:
    storage: 20Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: "/mnt/data"

```

---

```

apiVersion: v1
kind: PersistentVolumeClaim

```

```

metadata:
  name: mysql-pv-claim
spec:
  storageClassName: manual
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 20Gi

```

1. Deploy the PV and PVC of the YAML file:

```
kubectl apply -f https://k8s.io/examples/application/mysql/
mysql-pv.yaml
```

2. Deploy the contents of the YAML file:

```
kubectl apply -f https://k8s.io/examples/application/mysql/
mysql-deployment.yaml
```

3. Display information about the Deployment:

```

kubectl describe deployment mysql

Name: mysql
Namespace: default
CreationTimestamp: Tue, 01 Nov 2016 11:18:45 -0700
Labels: app=mysql
Annotations: deployment.kubernetes.io/revision=1
Selector: app=mysql
Replicas: 1 desired | 1 updated | 1 total | 0
available | 1 unavailable
StrategyType: Recreate
MinReadySeconds: 0
Pod Template:
  Labels: app=mysql
  Containers:
    mysql:
      Image: mysql:5.6
      Port: 3306/TCP
      Environment:
        MYSQL_ROOT_PASSWORD: password
      Mounts:
        /var/lib/mysql from mysql-persistent-storage (rw)
  Volumes:
    mysql-persistent-storage:
      Type: PersistentVolumeClaim (a reference to a
PersistentVolumeClaim in the same namespace)
      ClaimName: mysql-pv-claim
      ReadOnly: false
  Conditions:
    Type Status Reason
    ---- -----

```

```

Available      False   MinimumReplicasUnavailable
Progressing    True    ReplicaSetUpdated
OldReplicaSets: <none>
NewReplicaSet:  mysql-63082529 (1/1 replicas created)
Events:
  FirstSeen     LastSeen   Count  From
SubobjectPath  Type      Reason           Message
  -----        -----   -----  -----
  33s          33s       1      {deployment-
controller }           Normal           ScalingReplicaSet
Scaled up replica set mysql-63082529 to 1

```

4. List the pods created by the Deployment:

```
kubectl get pods -l app=mysql
```

NAME	READY	STATUS	RESTARTS	AGE
mysql-63082529-2z3ki	1/1	Running	0	3m

5. Inspect the PersistentVolumeClaim:

```
kubectl describe pvc mysql-pv-claim
```

```

Name:          mysql-pv-claim
Namespace:    default
StorageClass:
Status:        Bound
Volume:        mysql-pv-volume
Labels:        <none>
Annotations:   pv.kubernetes.io/bind-completed=yes
                pv.kubernetes.io/bound-by-controller=yes
Capacity:     20Gi
Access Modes: RWO
Events:        <none>

```

## **Accessing the MySQL instance**

The preceding YAML file creates a service that allows other Pods in the cluster to access the database. The Service option `clusterIP: None` lets the Service DNS name resolve directly to the Pod's IP address. This is optimal when you have only one Pod behind a Service and you don't intend to increase the number of Pods.

Run a MySQL client to connect to the server:

```
kubectl run -it --rm --image=mysql:5.6 --restart=Never mysql-client -- mysql -h mysql -ppassword
```

This command creates a new Pod in the cluster running a MySQL client and connects it to the server through the Service. If it connects, you know your stateful MySQL database is up and running.

```
Waiting for pod default/mysql-client-274442439-zyp6i to be
running, status is Pending, pod ready: false
If you don't see a command prompt, try pressing enter.
```

```
mysql>
```

## Updating

The image or any other part of the Deployment can be updated as usual with the `kubectl apply` command. Here are some precautions that are specific to stateful apps:

- Don't scale the app. This setup is for single-instance apps only. The underlying PersistentVolume can only be mounted to one Pod. For clustered stateful apps, see the [StatefulSet documentation](#).
- Use `strategy: type: Recreate` in the Deployment configuration YAML file. This instructs Kubernetes to not use rolling updates. Rolling updates will not work, as you cannot have more than one Pod running at a time. The `Recreate` strategy will stop the first pod before creating a new one with the updated configuration.

## Deleting a deployment

Delete the deployed objects by name:

```
kubectl delete deployment,svc mysql
kubectl delete pvc mysql-pv-claim
kubectl delete pv mysql-pv-volume
```

If you manually provisioned a PersistentVolume, you also need to manually delete it, as well as release the underlying resource. If you used a dynamic provisioner, it automatically deletes the PersistentVolume when it sees that you deleted the PersistentVolumeClaim. Some dynamic provisioners (such as those for EBS and PD) also release the underlying resource upon deleting the PersistentVolume.

## What's next

- Learn more about [Deployment objects](#).
- Learn more about [Deploying applications](#)
- [kubectl run documentation](#)
- [Volumes](#) and [Persistent Volumes](#)

## 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](#).

Last modified May 30, 2020 at 3:10 PM PST: [add en pages \(ecc27bbbe\)](#)  
[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Objectives](#)
- [Before you begin](#)
- [Deploy MySQL](#)
- [Accessing the MySQL instance](#)
- [Updating](#)
- [Deleting a deployment](#)
- [What's next](#)

# Run a Replicated Stateful Application

This page shows how to run a replicated stateful application using a [StatefulSet](#) controller. This application is a replicated MySQL database. The example topology has a single primary server and multiple replicas, using asynchronous row-based replication.

**Note: This is not a production configuration.** MySQL settings remain on insecure defaults to keep the focus on general patterns for running stateful applications in Kubernetes.

## Before you begin

- You need to have a Kubernetes cluster, and the `kubectl` command-line tool must be configured to communicate with your cluster. If you do not already have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:

- [Katacoda](#)
- [Play with Kubernetes](#)

To check the version, enter `kubectl version`.

- You need to either have a dynamic PersistentVolume provisioner with a default [StorageClass](#), or [statically provision PersistentVolumes](#) yourself to satisfy the [PersistentVolumeClaims](#) used here.
- This tutorial assumes you are familiar with [PersistentVolumes](#) and [StatefulSets](#), as well as other core concepts like [Pods](#), [Services](#), and [ConfigMaps](#).
- Some familiarity with MySQL helps, but this tutorial aims to present general patterns that should be useful for other systems.

# Objectives

- Deploy a replicated MySQL topology with a StatefulSet controller.
- Send MySQL client traffic.
- Observe resistance to downtime.
- Scale the StatefulSet up and down.

## Deploy MySQL

The example MySQL deployment consists of a ConfigMap, two Services, and a StatefulSet.

### ConfigMap

Create the ConfigMap from the following YAML configuration file:

[application/mysql/mysql-configmap.yaml](https://k8s.io/examples/application/mysql/mysql-configmap.yaml)  


```
apiVersion: v1
kind: ConfigMap
metadata:
  name: mysql
  labels:
    app: mysql
data:
  primary.cnf: |
    # Apply this config only on the primary.
    [mysqld]
    log-bin
  replica.cnf: |
    # Apply this config only on replicas.
    [mysqld]
    super-read-only
```

```
kubectl apply -f https://k8s.io/examples/application/mysql/mysql-
configmap.yaml
```

This ConfigMap provides `my.cnf` overrides that let you independently control configuration on the primary MySQL server and replicas. In this case, you want the primary server to be able to serve replication logs to replicas and you want replicas to reject any writes that don't come via replication.

There's nothing special about the ConfigMap itself that causes different portions to apply to different Pods. Each Pod decides which portion to look at as it's initializing, based on information provided by the StatefulSet controller.

## Services

Create the Services from the following YAML configuration file:

[application/mysql/mysql-services.yaml](#)

```
# Headless service for stable DNS entries of StatefulSet members.
apiVersion: v1
kind: Service
metadata:
  name: mysql
  labels:
    app: mysql
spec:
  ports:
  - name: mysql
    port: 3306
  clusterIP: None
  selector:
    app: mysql

# Client service for connecting to any MySQL instance for reads.
# For writes, you must instead connect to the primary:
mysql-read
apiVersion: v1
kind: Service
metadata:
  name: mysql-read
  labels:
    app: mysql
spec:
  ports:
  - name: mysql
    port: 3306
  selector:
    app: mysql
```

```
kubectl apply -f https://k8s.io/examples/application/mysql/mysql-services.yaml
```

The Headless Service provides a home for the DNS entries that the StatefulSet controller creates for each Pod that's part of the set. Because the Headless Service is named `mysql`, the Pods are accessible by resolving `<pod-name>.mysql` from within any other Pod in the same Kubernetes cluster and namespace.

The Client Service, called `mysql-read`, is a normal Service with its own cluster IP that distributes connections across all MySQL Pods that report being Ready. The set of potential endpoints includes the primary MySQL server and all replicas.

Note that only read queries can use the load-balanced Client Service. Because there is only one primary MySQL server, clients should connect directly to the primary MySQL Pod (through its DNS entry within the Headless Service) to execute writes.

## StatefulSet

Finally, create the StatefulSet from the following YAML configuration file:

[application/mysql/mysql-statefulset.yaml](#)  


```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: mysql
spec:
  selector:
    matchLabels:
      app: mysql
  serviceName: mysql
  replicas: 3
  template:
    metadata:
      labels:
        app: mysql
    spec:
      initContainers:
        - name: init-mysql
          image: mysql:5.7
          command:
            - bash
            - "-c"
            - |
              set -ex
              # Generate mysql server-id from pod ordinal index.
              [[ `hostname` =~ -([0-9]+)$ ]] || exit 1
              ordinal=${BASH_REMATCH[1]}
              echo [mysqld] > /mnt/conf.d/server-id.cnf
              # Add an offset to avoid reserved server-id=0 value.
              echo server-id=$((100 + $ordinal)) >> /mnt/conf.d/
server-id.cnf
          # Copy appropriate conf.d files from config-map to
emptyDir.
          if [[ $ordinal -eq 0 ]]; then
            cp /mnt/config-map/primary.cnf /mnt/conf.d/
          else
            cp /mnt/config-map/replica.cnf /mnt/conf.d/
          fi
      volumeMounts:
        - name: conf
```

```

    mountPath: /mnt/conf.d
  - name: config-map
    mountPath: /mnt/config-map
  - name: clone-mysql
    image: gcr.io/google-samples/xtrabackup:1.0
    command:
      - bash
      - "-c"
      - |
        set -ex
        # Skip the clone if data already exists.
        [[ -d /var/lib/mysql/mysql ]] && exit 0
        # Skip the clone on primary (ordinal index 0).
        [[ `hostname` =~ -([0-9]+)$ ]] || exit 1
        ordinal=${BASH_REMATCH[1]}
        [[ $ordinal -eq 0 ]] && exit 0
        # Clone data from previous peer.
        ncat --recv-only mysql-$((ordinal-1)).mysql 3307 | xbstream -x -C /var/lib/mysql
        # Prepare the backup.
        xtrabackup --prepare --target-dir=/var/lib/mysql
  volumeMounts:
    - name: data
      mountPath: /var/lib/mysql
      subPath: mysql
    - name: conf
      mountPath: /etc/mysql/conf.d
  containers:
    - name: mysql
      image: mysql:5.7
      env:
        - name: MYSQL_ALLOW_EMPTY_PASSWORD
          value: "1"
      ports:
        - name: mysql
          containerPort: 3306
  volumeMounts:
    - name: data
      mountPath: /var/lib/mysql
      subPath: mysql
    - name: conf
      mountPath: /etc/mysql/conf.d
  resources:
    requests:
      cpu: 500m
      memory: 1Gi
  livenessProbe:
    exec:
      command: ["mysqladmin", "ping"]
    initialDelaySeconds: 30
    periodSeconds: 10
    timeoutSeconds: 5

```

```

    readinessProbe:
      exec:
        # Check we can execute queries over TCP (skip-
networking is off).
        command: ["mysql", "-h", "127.0.0.1", "-e", "SELECT
1"]
          initialDelaySeconds: 5
          periodSeconds: 2
          timeoutSeconds: 1
        - name: xtrabackup
          image: gcr.io/google-samples/xtrabackup:1.0
          ports:
            - name: xtrabackup
              containerPort: 3307
              command:
                - bash
                - "-c"
                - |
                  set -ex
                  cd /var/lib/mysql

# Determine binlog position of cloned data, if any.
if [[ -f xtrabackup_slave_info && "x$(
<xtrabackup_slave_info)" != "x" ]]; then
  # XtraBackup already generated a partial "CHANGE
MASTER TO" query
  # because we're cloning from an existing replica.
  # (Need to remove the tailing semicolon!)
  cat xtrabackup_slave_info | sed -E 's/;$/;/g' > change
_master_to.sql.in
  # Ignore xtrabackup_binlog_info in this case (it's
useless).
  rm -f xtrabackup_slave_info xtrabackup_binlog_info
  elif [[ -f xtrabackup_binlog_info ]]; then
    # We're cloning directly from primary. Parse binlog
position.
    [[ `cat xtrabackup_binlog_info` =~ ^(.*)[:space:]+
(.*)$ ]] || exit 1
    rm -f xtrabackup_binlog_info xtrabackup_slave_info
    echo "CHANGE MASTER TO MASTER_LOG_FILE='\$
{BASH_REMATCH[1]}', \
          MASTER_LOG_POS=${BASH_REMATCH[2]}" > change_mas
ter_to.sql.in
    fi

# Check if we need to complete a clone by starting
replication.
if [[ -f change_master_to.sql.in ]]; then
  echo "Waiting for mysqld to be ready (accepting
connections)"
  until mysql -h 127.0.0.1 -e "SELECT 1"; do sleep 1; d
one

```

```

echo "Initializing replication from clone position"
mysql -h 127.0.0.1 \
-e "$(cat change_master_to.sql.in), \
MASTER_HOST='mysql-0.mysql', \
MASTER_USER='root', \
MASTER_PASSWORD='', \
MASTER_CONNECT_RETRY=10; \
START SLAVE;" || exit 1
# In case of container restart, attempt this at-most-
once.
mv change_master_to.sql.in change_master_to.sql.orig
fi

# Start a server to send backups when requested by
peers.
exec ncat --listen --keep-open --send-only --max-
conns=1 3307 -c \
"xtrabackup --backup --slave-info --stream=xbstream
--host=127.0.0.1 --user=root"
volumeMounts:
- name: data
  mountPath: /var/lib/mysql
  subPath: mysql
- name: conf
  mountPath: /etc/mysql/conf.d
resources:
  requests:
    cpu: 100m
    memory: 100Mi
volumes:
- name: conf
  emptyDir: {}
- name: config-map
  configMap:
    name: mysql
volumeClaimTemplates:
- metadata:
    name: data
spec:
  accessModes: ["ReadWriteOnce"]
  resources:
    requests:
      storage: 10Gi

```

```
kubectl apply -f https://k8s.io/examples/application/mysql/mysql-
statefulset.yaml
```

You can watch the startup progress by running:

```
kubectl get pods -l app=mysql --watch
```

After a while, you should see all 3 Pods become Running:

NAME	READY	STATUS	RESTARTS	AGE
mysql-0	2/2	Running	0	2m
mysql-1	2/2	Running	0	1m
mysql-2	2/2	Running	0	1m

Press **Ctrl+C** to cancel the watch. If you don't see any progress, make sure you have a dynamic PersistentVolume provisioner enabled as mentioned in the [prerequisites](#).

This manifest uses a variety of techniques for managing stateful Pods as part of a StatefulSet. The next section highlights some of these techniques to explain what happens as the StatefulSet creates Pods.

## Understanding stateful Pod initialization

The StatefulSet controller starts Pods one at a time, in order by their ordinal index. It waits until each Pod reports being Ready before starting the next one.

In addition, the controller assigns each Pod a unique, stable name of the form <statefulset-name>-<ordinal-index>, which results in Pods named mysql-0, mysql-1, and mysql-2.

The Pod template in the above StatefulSet manifest takes advantage of these properties to perform orderly startup of MySQL replication.

## Generating configuration

Before starting any of the containers in the Pod spec, the Pod first runs any [Init Containers](#) in the order defined.

The first Init Container, named `init-mysql`, generates special MySQL config files based on the ordinal index.

The script determines its own ordinal index by extracting it from the end of the Pod name, which is returned by the `hostname` command. Then it saves the ordinal (with a numeric offset to avoid reserved values) into a file called `server-id.cnf` in the MySQL `conf.d` directory. This translates the unique, stable identity provided by the StatefulSet controller into the domain of MySQL server IDs, which require the same properties.

The script in the `init-mysql` container also applies either `primary.cnf` or `replica.cnf` from the ConfigMap by copying the contents into `conf.d`.

Because the example topology consists of a single primary MySQL server and any number of replicas, the script simply assigns ordinal 0 to be the primary server, and everyone else to be replicas. Combined with the StatefulSet controller's [deployment order guarantee](#), this ensures the primary MySQL server is Ready before creating replicas, so they can begin replicating.

## **Cloning existing data**

*In general, when a new Pod joins the set as a replica, it must assume the primary MySQL server might already have data on it. It also must assume that the replication logs might not go all the way back to the beginning of time. These conservative assumptions are the key to allow a running StatefulSet to scale up and down over time, rather than being fixed at its initial size.*

*The second Init Container, named `clone-mysql`, performs a `clone` operation on a replica Pod the first time it starts up on an empty PersistentVolume. That means it copies all existing data from another running Pod, so its local state is consistent enough to begin replicating from the primary server.*

*MySQL itself does not provide a mechanism to do this, so the example uses a popular open-source tool called Percona XtraBackup. During the `clone`, the source MySQL server might suffer reduced performance. To minimize impact on the primary MySQL server, the script instructs each Pod to clone from the Pod whose ordinal index is one lower. This works because the StatefulSet controller always ensures Pod N is Ready before starting Pod N+1.*

## **Starting replication**

*After the Init Containers complete successfully, the regular containers run. The MySQL Pods consist of a `mysql` container that runs the actual `mysqld` server, and an `xtrabackup` container that acts as a [sidecar](#).*

*The `xtrabackup` sidecar looks at the cloned data files and determines if it's necessary to initialize MySQL replication on the replica. If so, it waits for `mysqld` to be ready and then executes the `CHANGE MASTER TO` and `START SLAVE` commands with replication parameters extracted from the XtraBackup clone files.*

*Once a replica begins replication, it remembers its primary MySQL server and reconnects automatically if the server restarts or the connection dies. Also, because replicas look for the primary server at its stable DNS name (`mysql-0.mysql`), they automatically find the primary server even if it gets a new Pod IP due to being rescheduled.*

*Lastly, after starting replication, the `xtrabackup` container listens for connections from other Pods requesting a data clone. This server remains up indefinitely in case the StatefulSet scales up, or in case the next Pod loses its PersistentVolumeClaim and needs to redo the clone.*

## **Sending client traffic**

*You can send test queries to the primary MySQL server (hostname `mysql-0.mysql`) by running a temporary container with the `mysql:5.7` image and running the `mysql` client binary.*

```
kubectl run mysql-client --image=mysql:5.7 -i --rm --restart=Never -- \
  mysql -h mysql-0.mysql <<EOF
CREATE DATABASE test;
CREATE TABLE test.messages (message VARCHAR(250));
INSERT INTO test.messages VALUES ('hello');
EOF
```

Use the hostname `mysql-read` to send test queries to any server that reports being Ready:

```
kubectl run mysql-client --image=mysql:5.7 -i -t --rm --restart=Never -- \
  mysql -h mysql-read -e "SELECT * FROM test.messages"
```

You should get output like this:

```
Waiting for pod default/mysql-client to be running, status is
Pending, pod ready: false
+-----+
| message |
+-----+
| hello   |
+-----+
pod "mysql-client" deleted
```

To demonstrate that the `mysql-read` Service distributes connections across servers, you can run `SELECT @@server_id` in a loop:

```
kubectl run mysql-client-loop --image=mysql:5.7 -i -t --rm --
restart=Never -- \
  bash -ic "while sleep 1; do mysql -h mysql-read -e 'SELECT
@@server_id,NOW()'; done"
```

You should see the reported `@@server_id` change randomly, because a different endpoint might be selected upon each connection attempt:

```
+-----+-----+
| @@server_id | NOW()           |
+-----+-----+
|      100  | 2006-01-02 15:04:05 |
+-----+-----+
| @@server_id | NOW()           |
+-----+-----+
|      102  | 2006-01-02 15:04:06 |
+-----+-----+
| @@server_id | NOW()           |
+-----+-----+
|      101  | 2006-01-02 15:04:07 |
+-----+-----+
```

You can press ***Ctrl+C*** when you want to stop the loop, but it's useful to keep it running in another window so you can see the effects of the following steps.

## **Simulating Pod and Node downtime**

To demonstrate the increased availability of reading from the pool of replicas instead of a single server, keep the `SELECT @@server_id` loop from above running while you force a Pod out of the Ready state.

### **Break the Readiness Probe**

The [readiness probe](#) for the `mysql` container runs the command `mysql -h 127.0.0.1 -e 'SELECT 1'` to make sure the server is up and able to execute queries.

One way to force this readiness probe to fail is to break that command:

```
kubectl exec mysql-2 -c mysql -- mv /usr/bin/mysql /usr/bin/mysql.off
```

This reaches into the actual container's filesystem for Pod `mysql-2` and renames the `mysql` command so the readiness probe can't find it. After a few seconds, the Pod should report one of its containers as not Ready, which you can check by running:

```
kubectl get pod mysql-2
```

Look for 1/2 in the `READY` column:

NAME	READY	STATUS	RESTARTS	AGE
mysql-2	1/2	Running	0	3m

At this point, you should see your `SELECT @@server_id` loop continue to run, although it never reports 102 anymore. Recall that the `init-mysql` script defined `server-id` as `100 + $ordinal`, so server ID 102 corresponds to Pod `mysql-2`.

Now repair the Pod and it should reappear in the loop output after a few seconds:

```
kubectl exec mysql-2 -c mysql -- mv /usr/bin/mysql.off /usr/bin/mysql
```

### **Delete Pods**

The `StatefulSet` also recreates Pods if they're deleted, similar to what a `ReplicaSet` does for stateless Pods.

```
kubectl delete pod mysql-2
```

The StatefulSet controller notices that no mysql-2 Pod exists anymore, and creates a new one with the same name and linked to the same PersistentVolumeClaim. You should see server ID 102 disappear from the loop output for a while and then return on its own.

## Drain a Node

If your Kubernetes cluster has multiple Nodes, you can simulate Node downtime (such as when Nodes are upgraded) by issuing a [drain](#).

First determine which Node one of the MySQL Pods is on:

```
kubectl get pod mysql-2 -o wide
```

The Node name should show up in the last column:

NAME	READY	STATUS	RESTARTS	AGE	IP
IP		NODE			
mysql-2	2/2	Running	0	15m	10.244.5.27
kubernetes-node-9l2t					

Then drain the Node by running the following command, which cordons it so no new Pods may schedule there, and then evicts any existing Pods. Replace <node-name> with the name of the Node you found in the last step.

This might impact other applications on the Node, so it's best to **only do this in a test cluster**.

```
kubectl drain <node-name> --force --delete-local-data --ignore-daemonsets
```

Now you can watch as the Pod reschedules on a different Node:

```
kubectl get pod mysql-2 -o wide --watch
```

It should look something like this:

NAME	READY	STATUS	RESTARTS	AGE
IP		NODE		
mysql-2	2/2	Terminating	0	15m
10.244.1.56		kubernetes-node-9l2t		
[...]				
mysql-2	0/2	Pending	0	0s
<none>		kubernetes-node-fjlm		
mysql-2	0/2	Init:0/2	0	0s
<none>		kubernetes-node-fjlm		
mysql-2	0/2	Init:1/2	0	20s
10.244.5.32		kubernetes-node-fjlm		
mysql-2	0/2	PodInitializing	0	21s
10.244.5.32		kubernetes-node-fjlm		
mysql-2	1/2	Running	0	22s
10.244.5.32		kubernetes-node-fjlm		

```
mysql-2  2/2    Running      0          30s
10.244.5.32 kubernetes-node-fjlm
```

And again, you should see server ID 102 disappear from the `SELECT @@server_id` loop output for a while and then return.

Now uncordon the Node to return it to a normal state:

```
kubectl uncordon <node-name>
```

## Scaling the number of replicas

With MySQL replication, you can scale your read query capacity by adding replicas. With StatefulSet, you can do this with a single command:

```
kubectl scale statefulset mysql --replicas=5
```

Watch the new Pods come up by running:

```
kubectl get pods -l app=mysql --watch
```

Once they're up, you should see server IDs 103 and 104 start appearing in the `SELECT @@server_id` loop output.

You can also verify that these new servers have the data you added before they existed:

```
kubectl run mysql-client --image=mysql:5.7 -i -t --rm --restart=Never -- \
  mysql -h mysql-3.mysql -e "SELECT * FROM test.messages"
```

```
Waiting for pod default/mysql-client to be running, status is
Pending, pod ready: false
+-----+
| message |
+-----+
| hello   |
+-----+
pod "mysql-client" deleted
```

Scaling back down is also seamless:

```
kubectl scale statefulset mysql --replicas=3
```

Note, however, that while scaling up creates new PersistentVolumeClaims automatically, scaling down does not automatically delete these PVCs. This gives you the choice to keep those initialized PVCs around to make scaling back up quicker, or to extract data before deleting them.

You can see this by running:

```
kubectl get pvc -l app=mysql
```

Which shows that all 5 PVCs still exist, despite having scaled the StatefulSet down to 3:

NAME	STATUS	VOLUME	CAPACITY
ACCESSMODES	AGE		
data-mysql-0	Bound	pvc-8acbf5dc-b103-11e6-93fa-42010a800002	10Gi RWO 20m
data-mysql-1	Bound	pvc-8ad39820-b103-11e6-93fa-42010a800002	10Gi RWO 20m
data-mysql-2	Bound	pvc-8ad69a6d-b103-11e6-93fa-42010a800002	10Gi RWO 20m
data-mysql-3	Bound	pvc-50043c45-b1c5-11e6-93fa-42010a800002	10Gi RWO 2m
data-mysql-4	Bound	pvc-500a9957-b1c5-11e6-93fa-42010a800002	10Gi RWO 2m

If you don't intend to reuse the extra PVCs, you can delete them:

```
kubectl delete pvc data-mysql-3  
kubectl delete pvc data-mysql-4
```

## Cleaning up

1. Cancel the `SELECT @@server_id` loop by pressing **Ctrl+C** in its terminal, or running the following from another terminal:

```
kubectl delete pod mysql-client-loop --now
```

2. Delete the StatefulSet. This also begins terminating the Pods.

```
kubectl delete statefulset mysql
```

3. Verify that the Pods disappear. They might take some time to finish terminating.

```
kubectl get pods -l app=mysql
```

You'll know the Pods have terminated when the above returns:

```
No resources found.
```

4. Delete the ConfigMap, Services, and PersistentVolumeClaims.

```
kubectl delete configmap,service,pvc -l app=mysql
```

5. If you manually provisioned PersistentVolumes, you also need to manually delete them, as well as release the underlying resources. If you used a dynamic provisioner, it automatically deletes the PersistentVolumes when it sees that you deleted the PersistentVolumeClaims. Some dynamic provisioners (such as those for EBS and PD) also release the underlying resources upon deleting the PersistentVolumes.

## What's next

- Learn more about [scaling a StatefulSet](#).
- Learn more about [debugging a StatefulSet](#).
- Learn more about [deleting a StatefulSet](#).
- Learn more about [force deleting StatefulSet Pods](#).
- Look in the [Helm Charts repository](#) for other stateful application examples.

## 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](#).

Last modified August 27, 2020 at 12:46 PM PST: [Incorporate suggested language change from Tim Bannister with a couple changes. \(b7f8f824b\)](#)  
[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Objectives](#)
- [Deploy MySQL](#)
  - [ConfigMap](#)
  - [Services](#)
  - [StatefulSet](#)
- [Understanding stateful Pod initialization](#)
  - [Generating configuration](#)
  - [Cloning existing data](#)
  - [Starting replication](#)
- [Sending client traffic](#)
- [Simulating Pod and Node downtime](#)
  - [Break the Readiness Probe](#)
  - [Delete Pods](#)
  - [Drain a Node](#)
- [Scaling the number of replicas](#)
- [Cleaning up](#)
- [What's next](#)

## Scale a StatefulSet

This task shows how to scale a StatefulSet. Scaling a StatefulSet refers to increasing or decreasing the number of replicas.

# **Before you begin**

- StatefulSets are only available in Kubernetes version 1.5 or later. To check your version of Kubernetes, run `kubectl version`.
- Not all stateful applications scale nicely. If you are unsure about whether to scale your StatefulSets, see [StatefulSet concepts](#) or [StatefulSet tutorial](#) for further information.
- You should perform scaling only when you are confident that your stateful application cluster is completely healthy.

# **Scaling StatefulSets**

## **Use kubectl to scale StatefulSets**

First, find the StatefulSet you want to scale.

```
kubectl get statefulsets <stateful-set-name>
```

Change the number of replicas of your StatefulSet:

```
kubectl scale statefulsets <stateful-set-name> --replicas=<new-replicas>
```

## **Make in-place updates on your StatefulSets**

Alternatively, you can do [in-place updates](#) on your StatefulSets.

If your StatefulSet was initially created with `kubectl apply`, update `.spec.replicas` of the StatefulSet manifests, and then do a `kubectl apply`:

```
kubectl apply -f <stateful-set-file-updated>
```

Otherwise, edit that field with `kubectl edit`:

```
kubectl edit statefulsets <stateful-set-name>
```

Or use `kubectl patch`:

```
kubectl patch statefulsets <stateful-set-name> -p '{"spec": {"replicas":<new-replicas>}}'
```

# **Troubleshooting**

## **Scaling down does not work right**

You cannot scale down a StatefulSet when any of the stateful Pods it manages is unhealthy. Scaling down only takes place after those stateful Pods become running and ready.

*If `spec.replicas > 1`, Kubernetes cannot determine the reason for an unhealthy Pod. It might be the result of a permanent fault or of a transient fault. A transient fault can be caused by a restart required by upgrading or maintenance.*

*If the Pod is unhealthy due to a permanent fault, scaling without correcting the fault may lead to a state where the StatefulSet membership drops below a certain minimum number of replicas that are needed to function correctly. This may cause your StatefulSet to become unavailable.*

*If the Pod is unhealthy due to a transient fault and the Pod might become available again, the transient error may interfere with your scale-up or scale-down operation. Some distributed databases have issues when nodes join and leave at the same time. It is better to reason about scaling operations at the application level in these cases, and perform scaling only when you are sure that your stateful application cluster is completely healthy.*

## What's next

- Learn more about [deleting a StatefulSet](#).

## 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](#).

Last modified May 30, 2020 at 3:10 PM PST: [add en pages \(ecc27bbbe\)](#)  
[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Scaling StatefulSets](#)
  - [Use kubectl to scale StatefulSets](#)
  - [Make in-place updates on your StatefulSets](#)
- [Troubleshooting](#)
  - [Scaling down does not work right](#)
- [What's next](#)

## Delete a StatefulSet

This task shows you how to delete a [StatefulSet](#).

## **Before you begin**

- This task assumes you have an application running on your cluster represented by a *StatefulSet*.

## **Deleting a StatefulSet**

You can delete a *StatefulSet* in the same way you delete other resources in Kubernetes: use the `kubectl delete` command, and specify the *StatefulSet* either by file or by name.

```
kubectl delete -f <file.yaml>
```

```
kubectl delete statefulsets <statefulset-name>
```

You may need to delete the associated headless service separately after the *StatefulSet* itself is deleted.

```
kubectl delete service <service-name>
```

Deleting a *StatefulSet* through `kubectl` will scale it down to 0, thereby deleting all pods that are a part of it. If you want to delete just the *StatefulSet* and not the pods, use `--cascade=false`.

```
kubectl delete -f <file.yaml> --cascade=false
```

By passing `--cascade=false` to `kubectl delete`, the Pods managed by the *StatefulSet* are left behind even after the *StatefulSet* object itself is deleted. If the pods have a label `app=myapp`, you can then delete them as follows:

```
kubectl delete pods -l app=myapp
```

## **Persistent Volumes**

Deleting the Pods in a *StatefulSet* will not delete the associated volumes. This is to ensure that you have the chance to copy data off the volume before deleting it. Deleting the PVC after the pods have terminated might trigger deletion of the backing Persistent Volumes depending on the storage class and reclaim policy. You should never assume ability to access a volume after claim deletion.

**Note:** Use caution when deleting a PVC, as it may lead to data loss.

## **Complete deletion of a StatefulSet**

To simply delete everything in a *StatefulSet*, including the associated pods, you can run a series of commands similar to the following:

```
grace=$(kubectl get pods <stateful-set-pod> --template '{{.spec.terminationGracePeriodSeconds}}')
```

```
kubectl delete statefulset -l app=myapp  
sleep $grace  
kubectl delete pvc -l app=myapp
```

In the example above, the Pods have the label `app=myapp`; substitute your own label as appropriate.

## Force deletion of StatefulSet pods

If you find that some pods in your StatefulSet are stuck in the 'Terminating' or 'Unknown' states for an extended period of time, you may need to manually intervene to forcefully delete the pods from the apiserver. This is a potentially dangerous task. Refer to [Force Delete StatefulSet Pods](#) for details.

## What's next

Learn more about [force deleting StatefulSet Pods](#).

## 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](#).

Last modified July 27, 2020 at 4:18 AM PST: [Revise Pod concept \(#22603\)](#) ([49eee8fd3](#))

[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Deleting a StatefulSet](#)
  - [Persistent Volumes](#)
  - [Complete deletion of a StatefulSet](#)
  - [Force deletion of StatefulSet pods](#)
- [What's next](#)

## Force Delete StatefulSet Pods

This page shows how to delete Pods which are part of a [stateful set](#), and explains the considerations to keep in mind when doing so.

# **Before you begin**

- This is a fairly advanced task and has the potential to violate some of the properties inherent to StatefulSet.
- Before proceeding, make yourself familiar with the considerations enumerated below.

## **StatefulSet considerations**

In normal operation of a StatefulSet, there is **never** a need to force delete a StatefulSet Pod. The [StatefulSet controller](#) is responsible for creating, scaling and deleting members of the StatefulSet. It tries to ensure that the specified number of Pods from ordinal 0 through N-1 are alive and ready. StatefulSet ensures that, at any time, there is at most one Pod with a given identity running in a cluster. This is referred to as at most one semantics provided by a StatefulSet.

Manual force deletion should be undertaken with caution, as it has the potential to violate the at most one semantics inherent to StatefulSet. StatefulSets may be used to run distributed and clustered applications which have a need for a stable network identity and stable storage. These applications often have configuration which relies on an ensemble of a fixed number of members with fixed identities. Having multiple members with the same identity can be disastrous and may lead to data loss (e.g. split brain scenario in quorum-based systems).

## **Delete Pods**

You can perform a graceful pod deletion with the following command:

```
kubectl delete pods <pod>
```

For the above to lead to graceful termination, the Pod **must not** specify a `pod.Spec.TerminationGracePeriodSeconds` of 0. The practice of setting a `pod.Spec.TerminationGracePeriodSeconds` of 0 seconds is unsafe and strongly discouraged for StatefulSet Pods. Graceful deletion is safe and will ensure that the Pod [shuts down gracefully](#) before the kubelet deletes the name from the apiserver.

Kubernetes (versions 1.5 or newer) will not delete Pods just because a Node is unreachable. The Pods running on an unreachable Node enter the 'Terminating' or 'Unknown' state after a [timeout](#). Pods may also enter these states when the user attempts graceful deletion of a Pod on an unreachable Node. The only ways in which a Pod in such a state can be removed from the apiserver are as follows:

- The Node object is deleted (either by you, or by the [Node Controller](#)).
- The kubelet on the unresponsive Node starts responding, kills the Pod and removes the entry from the apiserver.
- Force deletion of the Pod by the user.

The recommended best practice is to use the first or second approach. If a Node is confirmed to be dead (e.g. permanently disconnected from the network, powered down, etc), then delete the Node object. If the Node is suffering from a network partition, then try to resolve this or wait for it to resolve. When the partition heals, the kubelet will complete the deletion of the Pod and free up its name in the apiserver.

Normally, the system completes the deletion once the Pod is no longer running on a Node, or the Node is deleted by an administrator. You may override this by force deleting the Pod.

## Force Deletion

Force deletions **do not** wait for confirmation from the kubelet that the Pod has been terminated. Irrespective of whether a force deletion is successful in killing a Pod, it will immediately free up the name from the apiserver. This would let the StatefulSet controller create a replacement Pod with that same identity; this can lead to the duplication of a still-running Pod, and if said Pod can still communicate with the other members of the StatefulSet, will violate the at most one semantics that StatefulSet is designed to guarantee.

When you force delete a StatefulSet pod, you are asserting that the Pod in question will never again make contact with other Pods in the StatefulSet and its name can be safely freed up for a replacement to be created.

If you want to delete a Pod forcibly using kubectl version  $\geq 1.5$ , do the following:

```
kubectl delete pods <pod> --grace-period=0 --force
```

If you're using any version of kubectl  $\leq 1.4$ , you should omit the `--force` option and use:

```
kubectl delete pods <pod> --grace-period=0
```

If even after these commands the pod is stuck on Unknown state, use the following command to remove the pod from the cluster:

```
kubectl patch pod <pod> -p '{"metadata":{"finalizers":null}}'
```

Always perform force deletion of StatefulSet Pods carefully and with complete knowledge of the risks involved.

## What's next

Learn more about [debugging a StatefulSet](#).

## 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](#).

Last modified August 07, 2020 at 4:46 PM PST: [Tune links in tasks section \(1/2\) \(b8541d212\)](#)

[Edit this page](#) [Create child page](#) [Create an issue](#)

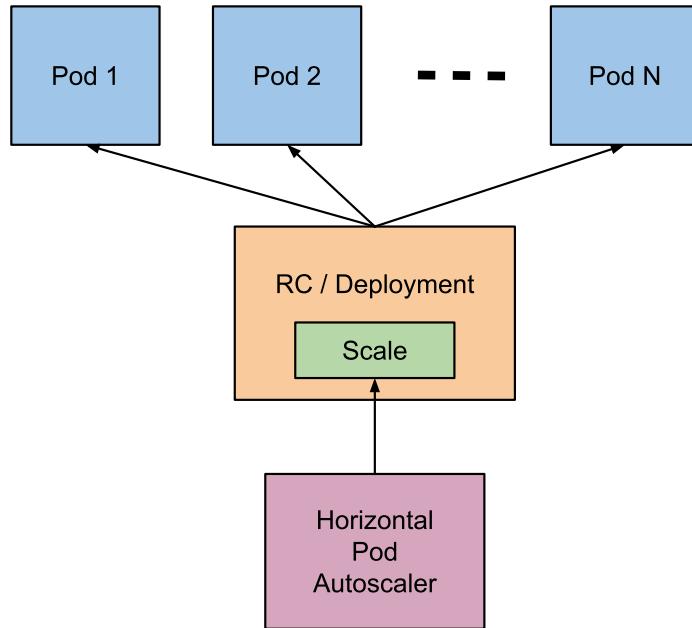
- [Before you begin](#)
- [StatefulSet considerations](#)
- [Delete Pods](#)
  - [Force Deletion](#)
- [What's next](#)

## Horizontal Pod Autoscaler

The Horizontal Pod Autoscaler automatically scales the number of Pods in a replication controller, deployment, replica set or stateful set based on observed CPU utilization (or, with [custom metrics](#) support, on some other application-provided metrics). Note that Horizontal Pod Autoscaling does not apply to objects that can't be scaled, for example, DaemonSets.

The Horizontal Pod Autoscaler is implemented as a Kubernetes API resource and a controller. The resource determines the behavior of the controller. The controller periodically adjusts the number of replicas in a replication controller or deployment to match the observed average CPU utilization to the target specified by user.

# **How does the Horizontal Pod Autoscaler work?**



*The Horizontal Pod Autoscaler is implemented as a control loop, with a period controlled by the controller manager's `--horizontal-pod-autoscaler-sync-period` flag (with a default value of 15 seconds).*

*During each period, the controller manager queries the resource utilization against the metrics specified in each `HorizontalPodAutoscaler` definition. The controller manager obtains the metrics from either the resource metrics API (for per-pod resource metrics), or the custom metrics API (for all other metrics).*

- For per-pod resource metrics (like CPU), the controller fetches the metrics from the resource metrics API for each Pod targeted by the `HorizontalPodAutoscaler`. Then, if a target utilization value is set, the controller calculates the utilization value as a percentage of the equivalent resource request on the containers in each Pod. If a target raw value is set, the raw metric values are used directly. The controller then takes the mean of the utilization or the raw value (depending on the type of target specified) across all targeted Pods, and produces a ratio used to scale the number of desired replicas.*

*Please note that if some of the Pod's containers do not have the relevant resource request set, CPU utilization for the Pod will not be*

defined and the autoscaler will not take any action for that metric. See the [algorithm details](#) section below for more information about how the autoscaling algorithm works.

- For per-pod custom metrics, the controller functions similarly to per-pod resource metrics, except that it works with raw values, not utilization values.
- For object metrics and external metrics, a single metric is fetched, which describes the object in question. This metric is compared to the target value, to produce a ratio as above. In the `autoscaling/v2beta2` API version, this value can optionally be divided by the number of Pods before the comparison is made.

The `HorizontalPodAutoscaler` normally fetches metrics from a series of aggregated APIs (`metrics.k8s.io`, `custom.metrics.k8s.io`, and `external.metrics.k8s.io`). The `metrics.k8s.io` API is usually provided by `metrics-server`, which needs to be launched separately. See [metrics-server](#) for instructions. The `HorizontalPodAutoscaler` can also fetch metrics directly from `Heapster`.

**Note:**

**FEATURE STATE:** Kubernetes v1.11 [deprecated]

Fetching metrics from `Heapster` is deprecated as of Kubernetes 1.11.

See [Support for metrics APIs](#) for more details.

The autoscaler accesses corresponding scalable controllers (such as replication controllers, deployments, and replica sets) by using the `scale` sub-resource. `Scale` is an interface that allows you to dynamically set the number of replicas and examine each of their current states. More details on `scale` sub-resource can be found [here](#).

## Algorithm Details

From the most basic perspective, the Horizontal Pod Autoscaler controller operates on the ratio between desired metric value and current metric value:

```
desiredReplicas = ceil[currentReplicas * ( currentMetricValue / desiredMetricValue )]
```

For example, if the current metric value is `200m`, and the desired value is `100m`, the number of replicas will be doubled, since  $200.0 / 100.0 == 2.0$ . If the current value is instead `50m`, we'll halve the number of replicas, since  $50.0 / 100.0 == 0.5$ . We'll skip scaling if the ratio is sufficiently close to `1.0` (within a globally-configurable tolerance, from the `--horizontal-pod-autoscaler-tolerance` flag, which defaults to `0.1`).

*When a `targetAverageValue` or `targetAverageUtilization` is specified, the `currentMetricValue` is computed by taking the average of the given metric across all Pods in the HorizontalPodAutoscaler's scale target. Before checking the tolerance and deciding on the final values, we take pod readiness and missing metrics into consideration, however.*

*All Pods with a deletion timestamp set (i.e. Pods in the process of being shut down) and all failed Pods are discarded.*

*If a particular Pod is missing metrics, it is set aside for later; Pods with missing metrics will be used to adjust the final scaling amount.*

*When scaling on CPU, if any pod has yet to become ready (i.e. it's still initializing) or the most recent metric point for the pod was before it became ready, that pod is set aside as well.*

*Due to technical constraints, the HorizontalPodAutoscaler controller cannot exactly determine the first time a pod becomes ready when determining whether to set aside certain CPU metrics. Instead, it considers a Pod "not yet ready" if it's unready and transitioned to unready within a short, configurable window of time since it started. This value is configured with the `--horizontal-pod-autoscaler-initial-readiness-delay` flag, and its default is 30 seconds. Once a pod has become ready, it considers any transition to ready to be the first if it occurred within a longer, configurable time since it started. This value is configured with the `--horizontal-pod-autoscaler-cpu-initialization-period` flag, and its default is 5 minutes.*

*The `currentMetricValue / desiredMetricValue` base scale ratio is then calculated using the remaining pods not set aside or discarded from above.*

*If there were any missing metrics, we recompute the average more conservatively, assuming those pods were consuming 100% of the desired value in case of a scale down, and 0% in case of a scale up. This dampens the magnitude of any potential scale.*

*Furthermore, if any not-yet-ready pods were present, and we would have scaled up without factoring in missing metrics or not-yet-ready pods, we conservatively assume the not-yet-ready pods are consuming 0% of the desired metric, further dampening the magnitude of a scale up.*

*After factoring in the not-yet-ready pods and missing metrics, we recalculate the usage ratio. If the new ratio reverses the scale direction, or is within the tolerance, we skip scaling. Otherwise, we use the new ratio to scale.*

*Note that the original value for the average utilization is reported back via the HorizontalPodAutoscaler status, without factoring in the not-yet-ready pods or missing metrics, even when the new usage ratio is used.*

*If multiple metrics are specified in a HorizontalPodAutoscaler, this calculation is done for each metric, and then the largest of the desired replica counts is chosen. If any of these metrics cannot be converted into a desired replica count (e.g. due to an error fetching the metrics from the metrics APIs) and a scale down is suggested by the metrics which can be*

fetched, scaling is skipped. This means that the HPA is still capable of scaling up if one or more metrics give a desiredReplicas greater than the current value.

Finally, just before HPA scales the target, the scale recommendation is recorded. The controller considers all recommendations within a configurable window choosing the highest recommendation from within that window. This value can be configured using the `--horizontal-pod-autoscaler-downscale-stabilization` flag, which defaults to 5 minutes. This means that scaledowns will occur gradually, smoothing out the impact of rapidly fluctuating metric values.

## **API Object**

The Horizontal Pod Autoscaler is an API resource in the Kubernetes `autoscaling` API group. The current stable version, which only includes support for CPU autoscaling, can be found in the `autoscaling/v1` API version.

The beta version, which includes support for scaling on memory and custom metrics, can be found in `autoscaling/v2beta2`. The new fields introduced in `autoscaling/v2beta2` are preserved as annotations when working with `autoscaling/v1`.

When you create a `HorizontalPodAutoscaler` API object, make sure the name specified is a valid [DNS subdomain name](#). More details about the API object can be found at [HorizontalPodAutoscaler Object](#).

## **Support for Horizontal Pod Autoscaler in kubectl**

Horizontal Pod Autoscaler, like every API resource, is supported in a standard way by `kubectl`. We can create a new autoscaler using `kubectl create` command. We can list autoscalers by `kubectl get hpa` and get detailed description by `kubectl describe hpa`. Finally, we can delete an autoscaler using `kubectl delete hpa`.

In addition, there is a special `kubectl autoscale` command for easy creation of a Horizontal Pod Autoscaler. For instance, executing `kubectl autoscale rs foo --min=2 --max=5 --cpu-percent=80` will create an autoscaler for replication set `foo`, with target CPU utilization set to 80% and the number of replicas between 2 and 5. The detailed documentation of `kubectl autoscale` can be found [here](#).

## **Autoscaling during rolling update**

Currently in Kubernetes, it is possible to perform a rolling update by using the deployment object, which manages the underlying replica sets for you. Horizontal Pod Autoscaler only supports the latter approach: the Horizontal Pod Autoscaler is bound to the deployment object, it sets the size for the

*deployment object, and the deployment is responsible for setting sizes of underlying replica sets.*

*Horizontal Pod Autoscaler does not work with rolling update using direct manipulation of replication controllers, i.e. you cannot bind a Horizontal Pod Autoscaler to a replication controller and do rolling update. The reason this doesn't work is that when rolling update creates a new replication controller, the Horizontal Pod Autoscaler will not be bound to the new replication controller.*

## **Support for cooldown/delay**

*When managing the scale of a group of replicas using the Horizontal Pod Autoscaler, it is possible that the number of replicas keeps fluctuating frequently due to the dynamic nature of the metrics evaluated. This is sometimes referred to as thrashing.*

*Starting from v1.6, a cluster operator can mitigate this problem by tuning the global HPA settings exposed as flags for the kube-controller-manager component:*

*Starting from v1.12, a new algorithmic update removes the need for the upscale delay.*

- **--horizontal-pod-autoscaler-downscale-stabilization:** The value for this option is a duration that specifies how long the autoscaler has to wait before another downscale operation can be performed after the current one has completed. The default value is 5 minutes (5m0s).

**Note:** When tuning these parameter values, a cluster operator should be aware of the possible consequences. If the delay (cooldown) value is set too long, there could be complaints that the Horizontal Pod Autoscaler is not responsive to workload changes. However, if the delay value is set too short, the scale of the replicas set may keep thrashing as usual.

## **Support for resource metrics**

*Any HPA target can be scaled based on the resource usage of the pods in the scaling target. When defining the pod specification the resource requests like cpu and memory should be specified. This is used to determine the resource utilization and used by the HPA controller to scale the target up or down. To use resource utilization based scaling specify a metric source like this:*

```
type: Resource
resource:
  name: cpu
  target:
    type: Utilization
    averageUtilization: 60
```

*With this metric the HPA controller will keep the average utilization of the pods in the scaling target at 60%. Utilization is the ratio between the current usage of resource to the requested resources of the pod. See [Algorithm](#) for more details about how the utilization is calculated and averaged.*

**Note:** Since the resource usages of all the containers are summed up the total pod utilization may not accurately represent the individual container resource usage. This could lead to situations where a single container might be running with high usage and the HPA will not scale out because the overall pod usage is still within acceptable limits.

## Container Resource Metrics

**FEATURE STATE:** Kubernetes v1.20 [alpha]

*HorizontalPodAutoscaler also supports a container metric source where the HPA can track the resource usage of individual containers across a set of Pods, in order to scale the target resource. This lets you configure scaling thresholds for the containers that matter most in a particular Pod. For example, if you have a web application and a logging sidecar, you can scale based on the resource use of the web application, ignoring the sidecar container and its resource use.*

*If you revise the target resource to have a new Pod specification with a different set of containers, you should revise the HPA spec if that newly added container should also be used for scaling. If the specified container in the metric source is not present or only present in a subset of the pods then those pods are ignored and the recommendation is recalculated. See [Algorithm](#) for more details about the calculation. To use container resources for autoscaling define a metric source as follows:*

```
type: ContainerResource
containerResource:
  name: cpu
  container: application
  target:
    type: Utilization
    averageUtilization: 60
```

*In the above example the HPA controller scales the target such that the average utilization of the cpu in the application container of all the pods is 60%.*

**Note:**

*If you change the name of a container that a HorizontalPodAutoscaler is tracking, you can make that change in a specific order to ensure scaling remains available and effective whilst the change is being applied. Before you update the resource that defines the container (such as a Deployment), you should*

*update the associated HPA to track both the new and old container names. This way, the HPA is able to calculate a scaling recommendation throughout the update process.*

*Once you have rolled out the container name change to the workload resource, tidy up by removing the old container name from the HPA specification.*

## **Support for multiple metrics**

*Kubernetes 1.6 adds support for scaling based on multiple metrics. You can use the `autoscaling/v2beta2` API version to specify multiple metrics for the Horizontal Pod Autoscaler to scale on. Then, the Horizontal Pod Autoscaler controller will evaluate each metric, and propose a new scale based on that metric. The largest of the proposed scales will be used as the new scale.*

## **Support for custom metrics**

**Note:** Kubernetes 1.2 added alpha support for scaling based on application-specific metrics using special annotations. Support for these annotations was removed in Kubernetes 1.6 in favor of the new autoscaling API. While the old method for collecting custom metrics is still available, these metrics will not be available for use by the Horizontal Pod Autoscaler, and the former annotations for specifying which custom metrics to scale on are no longer honored by the Horizontal Pod Autoscaler controller.

*Kubernetes 1.6 adds support for making use of custom metrics in the Horizontal Pod Autoscaler. You can add custom metrics for the Horizontal Pod Autoscaler to use in the `autoscaling/v2beta2` API. Kubernetes then queries the new custom metrics API to fetch the values of the appropriate custom metrics.*

*See [Support for metrics APIs](#) for the requirements.*

## **Support for metrics APIs**

*By default, the `HorizontalPodAutoscaler` controller retrieves metrics from a series of APIs. In order for it to access these APIs, cluster administrators must ensure that:*

- The [API aggregation layer](#) is enabled.
- The corresponding APIs are registered:
  - For resource metrics, this is the `metrics.k8s.io` API, generally provided by [metrics-server](#). It can be launched as a cluster addon.

- For custom metrics, this is the `custom.metrics.k8s.io` API. It's provided by "adapter" API servers provided by metrics solution vendors. Check with your metrics pipeline, or the [list of known solutions](#). If you would like to write your own, check out the [boilerplate](#) to get started.
- For external metrics, this is the `external.metrics.k8s.io` API. It may be provided by the custom metrics adapters provided above.
  - The `--horizontal-pod-autoscaler-use-rest-clients` is true or unset. Setting this to false switches to Heapster-based autoscaling, which is deprecated.

For more information on these different metrics paths and how they differ please see the relevant design proposals for [the HPA V2](#), [custom.metrics.k8s.io](#) and [external.metrics.k8s.io](#).

For examples of how to use them see [the walkthrough for using custom metrics](#) and [the walkthrough for using external metrics](#).

## **Support for configurable scaling behavior**

Starting from [v1.18](#) the v2beta2 API allows scaling behavior to be configured through the HPA `behavior` field. Behaviors are specified separately for scaling up and down in `scaleUp` or `scaleDown` section under the `behavior` field. A stabilization window can be specified for both directions which prevents the flapping of the number of the replicas in the scaling target. Similarly specifying scaling policies controls the rate of change of replicas while scaling.

### **Scaling Policies**

One or more scaling policies can be specified in the `behavior` section of the spec. When multiple policies are specified the policy which allows the highest amount of change is the policy which is selected by default. The following example shows this behavior while scaling down:

```
behavior:
  scaleDown:
    policies:
      - type: Pods
        value: 4
        periodSeconds: 60
      - type: Percent
        value: 10
        periodSeconds: 60
```

When the number of pods is more than 40 the second policy will be used for scaling down. For instance if there are 80 replicas and the target has to be scaled down to 10 replicas then during the first step 8 replicas will be reduced. In the next iteration when the number of replicas is 72, 10% of the pods is 7.2 but the number is rounded up to 8. On each loop of the

*autoscaler controller the number of pods to be change is re-calculated based on the number of current replicas. When the number of replicas falls below 40 the first policy (Pods) is applied and 4 replicas will be reduced at a time.*

*periodSeconds indicates the length of time in the past for which the policy must hold true. The first policy allows at most 4 replicas to be scaled down in one minute. The second policy allows at most 10% of the current replicas to be scaled down in one minute.*

*The policy selection can be changed by specifying the selectPolicy field for a scaling direction. By setting the value to Min which would select the policy which allows the smallest change in the replica count. Setting the value to Disabled completely disables scaling in that direction.*

## **Stabilization Window**

*The stabilization window is used to restrict the flapping of replicas when the metrics used for scaling keep fluctuating. The stabilization window is used by the autoscaling algorithm to consider the computed desired state from the past to prevent scaling. In the following example the stabilization window is specified for scaleDown.*

```
scaleDown:  
  stabilizationWindowSeconds: 300
```

*When the metrics indicate that the target should be scaled down the algorithm looks into previously computed desired states and uses the highest value from the specified interval. In above example all desired states from the past 5 minutes will be considered.*

## **Default Behavior**

*To use the custom scaling not all fields have to be specified. Only values which need to be customized can be specified. These custom values are merged with default values. The default values match the existing behavior in the HPA algorithm.*

```
behavior:  
  scaleDown:  
    stabilizationWindowSeconds: 300  
    policies:  
      - type: Percent  
        value: 100  
        periodSeconds: 15  
  scaleUp:  
    stabilizationWindowSeconds: 0  
    policies:  
      - type: Percent  
        value: 100  
        periodSeconds: 15  
      - type: Pods  
        value: 4
```

```
  periodSeconds: 15
  selectPolicy: Max
```

For scaling down the stabilization window is 300 seconds(or the value of the `--horizontal-pod-autoscaler-downscale-stabilization` flag if provided). There is only a single policy for scaling down which allows a 100% of the currently running replicas to be removed which means the scaling target can be scaled down to the minimum allowed replicas. For scaling up there is no stabilization window. When the metrics indicate that the target should be scaled up the target is scaled up immediately. There are 2 policies where 4 pods or a 100% of the currently running replicas will be added every 15 seconds till the HPA reaches its steady state.

## Example: change downscale stabilization window

To provide a custom downscale stabilization window of 1 minute, the following behavior would be added to the HPA:

```
behavior:
  scaleDown:
    stabilizationWindowSeconds: 60
```

## Example: limit scale down rate

To limit the rate at which pods are removed by the HPA to 10% per minute, the following behavior would be added to the HPA:

```
behavior:
  scaleDown:
    policies:
      - type: Percent
        value: 10
      periodSeconds: 60
```

To ensure that no more than 5 Pods are removed per minute, you can add a second scale-down policy with a fixed size of 5, and set `selectPolicy` to `minimum`. Setting `selectPolicy` to `Min` means that the autoscaler chooses the policy that affects the smallest number of Pods:

```
behavior:
  scaleDown:
    policies:
      - type: Percent
        value: 10
      periodSeconds: 60
      - type: Pods
        value: 5
      periodSeconds: 60
    selectPolicy: Min
```

## **Example: disable scale down**

The `selectPolicy` value of `Disabled` turns off scaling the given direction. So to prevent downscaling the following policy would be used:

```
behavior:  
  scaleDown:  
    selectPolicy: Disabled
```

## **Implicit maintenance-mode deactivation**

You can implicitly deactivate the HPA for a target without the need to change the HPA configuration itself. If the target's desired replica count is set to 0, and the HPA's minimum replica count is greater than 0, the HPA stops adjusting the target (and sets the `ScalingActive` Condition on itself to `false`) until you reactivate it by manually adjusting the target's desired replica count or HPA's minimum replica count.

## **What's next**

- Design documentation: [Horizontal Pod Autoscaling](#).
- `kubectl autoscale` command: [kubectl autoscale](#).
- Usage example of [Horizontal Pod Autoscaler](#).

## **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](#).

Last modified November 02, 2020 at 8:44 PM PST: [Added docs about container resource metric source for HPA \(#23523\) \(b838012e1\)](#)  
[Edit this page](#) [Create child page](#) [Create an issue](#)

- [How does the Horizontal Pod Autoscaler work?](#)
  - [Algorithm Details](#)
- [API Object](#)
- [Support for Horizontal Pod Autoscaler in kubectl](#)
- [Autoscaling during rolling update](#)
- [Support for cooldown/delay](#)
- [Support for resource metrics](#)
  - [Container Resource Metrics](#)
- [Support for multiple metrics](#)
- [Support for custom metrics](#)
- [Support for metrics APIs](#)

- [Support for configurable scaling behavior](#)
  - [Scaling Policies](#)
  - [Stabilization Window](#)
  - [Default Behavior](#)
  - [Example: change downscale stabilization window](#)
  - [Example: limit scale down rate](#)
  - [Example: disable scale down](#)
- [Implicit maintenance-mode deactivation](#)
- [What's next](#)

# **Horizontal Pod Autoscaler Walkthrough**

*Horizontal Pod Autoscaler automatically scales the number of Pods in a replication controller, deployment, replica set or stateful set based on observed CPU utilization (or, with beta support, on some other, application-provided metrics).*

*This document walks you through an example of enabling Horizontal Pod Autoscaler for the php-apache server. For more information on how Horizontal Pod Autoscaler behaves, see the [Horizontal Pod Autoscaler user guide](#).*

## **Before you begin**

*This example requires a running Kubernetes cluster and kubectl, version 1.2 or later. [Metrics server](#) monitoring needs to be deployed in the cluster to provide metrics through the [Metrics API](#). Horizontal Pod Autoscaler uses this API to collect metrics. To learn how to deploy the metrics-server, see the [metrics-server documentation](#).*

*To specify multiple resource metrics for a Horizontal Pod Autoscaler, you must have a Kubernetes cluster and kubectl at version 1.6 or later. To make use of custom metrics, your cluster must be able to communicate with the API server providing the custom Metrics API. Finally, to use metrics not related to any Kubernetes object you must have a Kubernetes cluster at version 1.10 or later, and you must be able to communicate with the API server that provides the external Metrics API. See the [Horizontal Pod Autoscaler user guide](#) for more details.*

## **Run and expose php-apache server**

*To demonstrate Horizontal Pod Autoscaler we will use a custom docker image based on the php-apache image. The Dockerfile has the following content:*

```
FROM php:5-apache
COPY index.php /var/www/html/index.php
RUN chmod a+rx index.php
```

It defines an `index.php` page which performs some CPU intensive computations:

```
<?php
$x = 0.0001;
for ($i = 0; $i <= 1000000; $i++) {
    $x += sqrt($x);
}
echo "OK!";
?>
```

First, we will start a deployment running the image and expose it as a service using the following configuration:

[application/php-apache.yaml](#)



```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: php-apache
spec:
  selector:
    matchLabels:
      run: php-apache
  replicas: 1
  template:
    metadata:
      labels:
        run: php-apache
    spec:
      containers:
        - name: php-apache
          image: k8s.gcr.io/hpa-example
          ports:
            - containerPort: 80
          resources:
            limits:
              cpu: 500m
            requests:
              cpu: 200m
---
apiVersion: v1
kind: Service
metadata:
  name: php-apache
  labels:
    run: php-apache
```

```
spec:  
  ports:  
    - port: 80  
  selector:  
    run: php-apache
```

Run the following command:

```
kubectl apply -f https://k8s.io/examples/application/php-apache.yaml
```

```
deployment.apps/php-apache created  
service/php-apache created
```

## Create Horizontal Pod Autoscaler

Now that the server is running, we will create the autoscaler using [kubectl autoscale](#). The following command will create a Horizontal Pod Autoscaler that maintains between 1 and 10 replicas of the Pods controlled by the `php-apache` deployment we created in the first step of these instructions. Roughly speaking, HPA will increase and decrease the number of replicas (via the deployment) to maintain an average CPU utilization across all Pods of 50% (since each pod requests 200 milli-cores by `kubectl run`), this means average CPU usage of 100 milli-cores). See [here](#) for more details on the algorithm.

```
kubectl autoscale deployment php-apache --cpu-percent=50 --min=1  
--max=10
```

```
horizontalpodautoscaler.autoscaling/php-apache autoscaled
```

We may check the current status of autoscaler by running:

```
kubectl get hpa
```

NAME	REFERENCE	TARGET	MINPODS
MAXPODS	REPLICAS	AGE	
php-apache	Deployment/php-apache/scale	0% / 50%	1
10	1	18s	

Please note that the current CPU consumption is 0% as we are not sending any requests to the server (the TARGET column shows the average across all the pods controlled by the corresponding deployment).

## Increase load

Now, we will see how the autoscaler reacts to increased load. We will start a container, and send an infinite loop of queries to the `php-apache` service (please run it in a different terminal):

```
kubectl run -i --tty load-generator --rm --image=busybox --restart=Never -- /bin/sh -c "while sleep 0.01; do wget -q -O http://php-apache; done"
```

Within a minute or so, we should see the higher CPU load by executing:

```
kubectl get hpa
```

NAME	REFERENCE	TARGET	MINPODS
MAXPODS	REPLICAS	AGE	
php-apache	Deployment/php-apache/scale	305% / 50%	1
10	1	3m	

Here, CPU consumption has increased to 305% of the request. As a result, the deployment was resized to 7 replicas:

```
kubectl get deployment php-apache
```

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
php-apache	7/7	7	7	19m

**Note:** It may take a few minutes to stabilize the number of replicas. Since the amount of load is not controlled in any way it may happen that the final number of replicas will differ from this example.

## Stop load

We will finish our example by stopping the user load.

In the terminal where we created the container with busybox image, terminate the load generation by typing <Ctrl> + C.

Then we will verify the result state (after a minute or so):

```
kubectl get hpa
```

NAME	REFERENCE	TARGET	
MINPODS	MAXPODS	REPLICAS	AGE
php-apache	Deployment/php-apache/scale	0% / 50%	
1	10	1	11m

```
kubectl get deployment php-apache
```

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
php-apache	1/1	1	1	27m

Here CPU utilization dropped to 0, and so HPA autoscaled the number of replicas back down to 1.

**Note:** Autoscaling the replicas may take a few minutes.

## **Autoscaling on multiple metrics and custom metrics**

You can introduce additional metrics to use when autoscaling the `php-apache` Deployment by making use of the `autoscaling/v2beta2` API version.

First, get the YAML of your `HorizontalPodAutoscaler` in the `autoscaling/v2beta2` form:

```
kubectl get hpa.v2beta2.autoscaling -o yaml > /tmp/hpa-v2.yaml
```

Open the `/tmp/hpa-v2.yaml` file in an editor, and you should see YAML which looks like this:

```
apiVersion: autoscaling/v2beta2
kind: HorizontalPodAutoscaler
metadata:
  name: php-apache
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: php-apache
  minReplicas: 1
  maxReplicas: 10
  metrics:
    - type: Resource
      resource:
        name: cpu
        target:
          type: Utilization
          averageUtilization: 50
  status:
    observedGeneration: 1
    lastScaleTime: <some-time>
    currentReplicas: 1
    desiredReplicas: 1
    currentMetrics:
      - type: Resource
        resource:
          name: cpu
          current:
            averageUtilization: 0
            averageValue: 0
```

Notice that the `targetCPUUtilizationPercentage` field has been replaced with an array called `metrics`. The CPU utilization metric is a resource metric, since it is represented as a percentage of a resource specified on pod containers. Notice that you can specify other resource metrics besides CPU. By default, the only other supported resource metric is memory. These

*resources do not change names from cluster to cluster, and should always be available, as long as the metrics.k8s.io API is available.*

*You can also specify resource metrics in terms of direct values, instead of as percentages of the requested value, by using a target.type of AverageValue instead of Utilization, and setting the corresponding target.averageValue field instead of the target.averageUtilization.*

*There are two other types of metrics, both of which are considered custom metrics: pod metrics and object metrics. These metrics may have names which are cluster specific, and require a more advanced cluster monitoring setup.*

*The first of these alternative metric types is pod metrics. These metrics describe Pods, and are averaged together across Pods and compared with a target value to determine the replica count. They work much like resource metrics, except that they only support a target type of AverageValue.*

*Pod metrics are specified using a metric block like this:*

```
type: Pods
pods:
  metric:
    name: packets-per-second
  target:
    type: AverageValue
    averageValue: 1k
```

*The second alternative metric type is object metrics. These metrics describe a different object in the same namespace, instead of describing Pods. The metrics are not necessarily fetched from the object; they only describe it. Object metrics support target types of both Value and AverageValue. With Value, the target is compared directly to the returned metric from the API. With AverageValue, the value returned from the custom metrics API is divided by the number of Pods before being compared to the target. The following example is the YAML representation of the requests-per-second metric.*

```
type: Object
object:
  metric:
    name: requests-per-second
  describedObject:
    apiVersion: networking.k8s.io/v1beta1
    kind: Ingress
    name: main-route
  target:
    type: Value
    value: 2k
```

*If you provide multiple such metric blocks, the HorizontalPodAutoscaler will consider each metric in turn. The HorizontalPodAutoscaler will calculate*

*proposed replica counts for each metric, and then choose the one with the highest replica count.*

*For example, if you had your monitoring system collecting metrics about network traffic, you could update the definition above using kubectl edit to look like this:*

```
apiVersion: autoscaling/v2beta2
kind: HorizontalPodAutoscaler
metadata:
  name: php-apache
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: php-apache
  minReplicas: 1
  maxReplicas: 10
  metrics:
    - type: Resource
      resource:
        name: cpu
        target:
          type: Utilization
          averageUtilization: 50
    - type: Pods
      pods:
        metric:
          name: packets-per-second
          target:
            type: AverageValue
            averageValue: 1k
    - type: Object
      object:
        metric:
          name: requests-per-second
        describedObject:
          apiVersion: networking.k8s.io/v1beta1
          kind: Ingress
          name: main-route
          target:
            type: Value
            value: 10k
  status:
    observedGeneration: 1
    lastScaleTime: <some-time>
    currentReplicas: 1
    desiredReplicas: 1
    currentMetrics:
      - type: Resource
        resource:
          name: cpu
```

```

  current: 0
  averageUtilization: 0
  averageValue: 0
- type: Object
  object:
    metric:
      name: requests-per-second
  describedObject:
    apiVersion: networking.k8s.io/v1beta1
    kind: Ingress
    name: main-route
  current:
    value: 10k

```

Then, your HorizontalPodAutoscaler would attempt to ensure that each pod was consuming roughly 50% of its requested CPU, serving 1000 packets per second, and that all pods behind the main-route Ingress were serving a total of 10000 requests per second.

## **Autoscaling on more specific metrics**

Many metrics pipelines allow you to describe metrics either by name or by a set of additional descriptors called *labels*. For all non-resource metric types (*pod*, *object*, and *external*, described below), you can specify an additional label selector which is passed to your metric pipeline. For instance, if you collect a metric *http\_requests* with the *verb* label, you can specify the following metric block to scale only on GET requests:

```

type: Object
object:
  metric:
    name: http_requests
    selector: {matchLabels: {verb: GET}}

```

This selector uses the same syntax as the full Kubernetes label selectors. The monitoring pipeline determines how to collapse multiple series into a single value, if the name and selector match multiple series. The selector is additive, and cannot select metrics that describe objects that are **not** the target object (the target pods in the case of the *Pods* type, and the described object in the case of the *Object* type).

## **Autoscaling on metrics not related to Kubernetes objects**

Applications running on Kubernetes may need to autoscale based on metrics that don't have an obvious relationship to any object in the Kubernetes cluster, such as metrics describing a hosted service with no direct correlation to Kubernetes namespaces. In Kubernetes 1.10 and later, you can address this use case with external metrics.

Using external metrics requires knowledge of your monitoring system; the setup is similar to that required when using custom metrics. External metrics allow you to autoscale your cluster based on any metric available in

your monitoring system. Just provide a `metric` block with a `name` and `select` or, as above, and use the `External` metric type instead of `Object`. If multiple time series are matched by the `metricSelector`, the sum of their values is used by the `HorizontalPodAutoscaler`. External metrics support both the `Value` and `AverageValue` target types, which function exactly the same as when you use the `Object` type.

For example if your application processes tasks from a hosted queue service, you could add the following section to your `HorizontalPodAutoscaler` manifest to specify that you need one worker per 30 outstanding tasks.

```
- type: External
  external:
    metric:
      name: queue_messages_ready
      selector: "queue=worker_tasks"
    target:
      type: AverageValue
      averageValue: 30
```

When possible, it's preferable to use the custom metric target types instead of external metrics, since it's easier for cluster administrators to secure the custom metrics API. The external metrics API potentially allows access to any metric, so cluster administrators should take care when exposing it.

## Appendix: Horizontal Pod Autoscaler Status Conditions

When using the `autoscaling/v2beta2` form of the `HorizontalPodAutoscaler`, you will be able to see status conditions set by Kubernetes on the `HorizontalPodAutoscaler`. These status conditions indicate whether or not the `HorizontalPodAutoscaler` is able to scale, and whether or not it is currently restricted in any way.

The conditions appear in the `status.conditions` field. To see the conditions affecting a `HorizontalPodAutoscaler`, we can use `kubectl describe hpa`:

```
kubectl describe hpa cm-test
```

Name:	cm-test
Namespace:	prom
Labels:	<none>
Annotations:	<none>
CreationTimestamp:	Fri, 16 Jun 2017 18:09:22 +0000
Reference:	ReplicationController/cm-test
Metrics:	( current / target )
"http_requests" on pods:	66m / 500m
Min replicas:	1
Max replicas:	4
ReplicationController pods:	1 current / 1 desired
Conditions:	

Type	Status	Reason	Message
AbleToScale	True	ReadyForNewScale	the last scale time was sufficiently old as to warrant a new scale
ScalingActive	True	ValidMetricFound	the HPA was able to successfully calculate a replica count from pods metric http_requests
ScalingLimited	False	DesiredWithinRange	the desired replica count is within the acceptable range

Events:

For this HorizontalPodAutoscaler, we can see several conditions in a healthy state. The first, `AbleToScale`, indicates whether or not the HPA is able to fetch and update scales, as well as whether or not any backoff-related conditions would prevent scaling. The second, `ScalingActive`, indicates whether or not the HPA is enabled (i.e. the replica count of the target is not zero) and is able to calculate desired scales. When it is `False`, it generally indicates problems with fetching metrics. Finally, the last condition, `Scaling Limited`, indicates that the desired scale was capped by the maximum or minimum of the HorizontalPodAutoscaler. This is an indication that you may wish to raise or lower the minimum or maximum replica count constraints on your HorizontalPodAutoscaler.

## Appendix: Quantities

All metrics in the HorizontalPodAutoscaler and metrics APIs are specified using a special whole-number notation known in Kubernetes as a [quantity](#). For example, the quantity `10500m` would be written as `10.5` in decimal notation. The metrics APIs will return whole numbers without a suffix when possible, and will generally return quantities in milli-units otherwise. This means you might see your metric value fluctuate between `1` and `1500m`, or `1` and `1.5` when written in decimal notation.

## Appendix: Other possible scenarios

### Creating the autoscaler declaratively

Instead of using `kubectl autoscale` command to create a HorizontalPodAutoscaler imperatively we can use the following file to create it declaratively:

[application/hpa/php-apache.yaml](#)  
□

```
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: php-apache
spec:
  scaleTargetRef:
```

```
apiVersion: apps/v1
kind: Deployment
  name: php-apache
  minReplicas: 1
  maxReplicas: 10
  targetCPUUtilizationPercentage: 50
```

We will create the autoscaler by executing the following command:

```
kubectl create -f https://k8s.io/examples/application/hpa/php-
apache.yaml
```

```
horizontalpodautoscaler.autoscaling/php-apache created
```

## 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](#).

Last modified October 28, 2020 at 11:19 AM PST: [clean up turnkey cloud solutions \(dd618cff3\)](#)

[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Run and expose php-apache server](#)
- [Create Horizontal Pod Autoscaler](#)
- [Increase load](#)
- [Stop load](#)
- [Autoscaling on multiple metrics and custom metrics](#)
  - [Autoscaling on more specific metrics](#)
  - [Autoscaling on metrics not related to Kubernetes objects](#)
- [Appendix: Horizontal Pod Autoscaler Status Conditions](#)
- [Appendix: Quantities](#)
- [Appendix: Other possible scenarios](#)
  - [Creating the autoscaler declaratively](#)

# Specifying a Disruption Budget for your Application

**FEATURE STATE:** Kubernetes v1.5 [beta]

This page shows how to limit the number of concurrent disruptions that your application experiences, allowing for higher availability while permitting the cluster administrator to manage the clusters nodes.

## **Before you begin**

- You are the owner of an application running on a Kubernetes cluster that requires high availability.
- You should know how to deploy [Replicated Stateless Applications](#) and/or [Replicated Stateful Applications](#).
- You should have read about [Pod Disruptions](#).
- You should confirm with your cluster owner or service provider that they respect Pod Disruption Budgets.

## **Protecting an Application with a PodDisruptionBudget**

1. Identify what application you want to protect with a `PodDisruptionBudget (PDB)`.
2. Think about how your application reacts to disruptions.
3. Create a PDB definition as a YAML file.
4. Create the PDB object from the YAML file.

### **Identify an Application to Protect**

The most common use case when you want to protect an application specified by one of the built-in Kubernetes controllers:

- Deployment
- ReplicationController
- ReplicaSet
- StatefulSet

In this case, make a note of the controller's `.spec.selector`; the same selector goes into the PDBs `.spec.selector`.

From version 1.15 PDBs support custom controllers where the [scale subresource](#) is enabled.

You can also use PDBs with pods which are not controlled by one of the above controllers, or arbitrary groups of pods, but there are some restrictions, described in [Arbitrary Controllers and Selectors](#).

### **Think about how your application reacts to disruptions**

Decide how many instances can be down at the same time for a short period due to a voluntary disruption.

- Stateless frontends:
  - Concern: don't reduce serving capacity by more than 10%.
  - Solution: use PDB with `minAvailable 90%` for example.

- *Single-instance Stateful Application:*
  - *Concern: do not terminate this application without talking to me.*
  - *Possible Solution 1: Do not use a PDB and tolerate occasional downtime.*
  - *Possible Solution 2: Set PDB with maxUnavailable=0. Have an understanding (outside of Kubernetes) that the cluster operator needs to consult you before termination. When the cluster operator contacts you, prepare for downtime, and then delete the PDB to indicate readiness for disruption. Recreate afterwards.*
- *Multiple-instance Stateful application such as Consul, ZooKeeper, or etcd:*
  - *Concern: Do not reduce number of instances below quorum, otherwise writes fail.*
  - *Possible Solution 1: set maxUnavailable to 1 (works with varying scale of application).*
  - *Possible Solution 2: set minAvailable to quorum-size (e.g. 3 when scale is 5). (Allows more disruptions at once).*
- *Restartable Batch Job:*
  - *Concern: Job needs to complete in case of voluntary disruption.*
  - *Possible solution: Do not create a PDB. The Job controller will create a replacement pod.*

## **Rounding logic when specifying percentages**

*Values for minAvailable or maxUnavailable can be expressed as integers or as a percentage.*

- *When you specify an integer, it represents a number of Pods. For instance, if you set minAvailable to 10, then 10 Pods must always be available, even during a disruption.*
- *When you specify a percentage by setting the value to a string representation of a percentage (eg. "50%"), it represents a percentage of total Pods. For instance, if you set maxUnavailable to "50%", then only 50% of the Pods can be unavailable during a disruption.*

*When you specify the value as a percentage, it may not map to an exact number of Pods. For example, if you have 7 Pods and you set minAvailable to "50%", it's not immediately obvious whether that means 3 Pods or 4 Pods must be available. Kubernetes rounds up to the nearest integer, so in this case, 4 Pods must be available. You can examine the [code](#) that controls this behavior.*

## **Specifying a PodDisruptionBudget**

*A PodDisruptionBudget has three fields:*

- *A label selector .spec.selector to specify the set of pods to which it applies. This field is required.*
- *.spec.minAvailable which is a description of the number of pods from that set that must still be available after the eviction, even in the*

*absence of the evicted pod. `minAvailable` can be either an absolute number or a percentage.*

- `.spec.maxUnavailable` (available in Kubernetes 1.7 and higher) which is a description of the number of pods from that set that can be unavailable after the eviction. It can be either an absolute number or a percentage.

**Note:** For versions 1.8 and earlier: When creating a `PodDisruptionBudget` object using the `kubectl` command line tool, the `minAvailable` field has a default value of 1 if neither `minAvailable` nor `maxUnavailable` is specified.

You can specify only one of `maxUnavailable` and `minAvailable` in a single `PodDisruptionBudget`. `maxUnavailable` can only be used to control the eviction of pods that have an associated controller managing them. In the examples below, "desired replicas" is the scale of the controller managing the pods being selected by the `PodDisruptionBudget`.

*Example 1: With a `minAvailable` of 5, evictions are allowed as long as they leave behind 5 or more healthy pods among those selected by the `PodDisruptionBudget`'s selector.*

*Example 2: With a `minAvailable` of 30%, evictions are allowed as long as at least 30% of the number of desired replicas are healthy.*

*Example 3: With a `maxUnavailable` of 5, evictions are allowed as long as there are at most 5 unhealthy replicas among the total number of desired replicas.*

*Example 4: With a `maxUnavailable` of 30%, evictions are allowed as long as no more than 30% of the desired replicas are unhealthy.*

*In typical usage, a single budget would be used for a collection of pods managed by a controller—for example, the pods in a single `ReplicaSet` or `StatefulSet`.*

**Note:** A disruption budget does not truly guarantee that the specified number/percentage of pods will always be up. For example, a node that hosts a pod from the collection may fail when the collection is at the minimum size specified in the budget, thus bringing the number of available pods from the collection below the specified size. The budget can only protect against voluntary evictions, not all causes of unavailability.

*If you set `maxUnavailable` to 0% or 0, or you set `minAvailable` to 100% or the number of replicas, you are requiring zero voluntary evictions. When you set zero voluntary evictions for a workload object such as `ReplicaSet`, then you cannot successfully drain a Node running one of those Pods. If you try to drain a Node where an unevictable Pod is running, the drain never completes. This is permitted as per the semantics of `PodDisruptionBudget`.*

*You can find examples of pod disruption budgets defined below. They match pods with the label `app: zookeeper`.*

Example PDB Using `minAvailable`:

[`policy/zookeeper-pod-disruption-budget-minavailable.yaml`](#)



```
apiVersion: policy/v1beta1
kind: PodDisruptionBudget
metadata:
  name: zk-pdb
spec:
  minAvailable: 2
  selector:
    matchLabels:
      app: zookeeper
```

Example PDB Using `maxUnavailable` (Kubernetes 1.7 or higher):

[`policy/zookeeper-pod-disruption-budget-maxunavailable.yaml`](#)



```
apiVersion: policy/v1beta1
kind: PodDisruptionBudget
metadata:
  name: zk-pdb
spec:
  maxUnavailable: 1
  selector:
    matchLabels:
      app: zookeeper
```

For example, if the above `zk-pdb` object selects the pods of a `StatefulSet` of size 3, both specifications have the exact same meaning. The use of `maxUnavailable` is recommended as it automatically responds to changes in the number of replicas of the corresponding controller.

## Create the PDB object

You can create or update the PDB object with a command like `kubectl apply -f mypdb.yaml`.

## Check the status of the PDB

Use `kubectl` to check that your PDB is created.

Assuming you don't actually have pods matching `app: zookeeper` in your namespace, then you'll see something like this:

```
kubectl get poddisruptionbudgets
```

NAME	MIN AVAILABLE	MAX UNAVAILABLE	ALLOWED DISRUPTIONS
AGE			

<code>zk-pdb</code>	2	<code>N/A</code>	0
	<code>7s</code>		

If there are matching pods (say, 3), then you would see something like this:

```
kubectl get poddisruptionbudgets
```

NAME	MIN AVAILABLE	MAX UNAVAILABLE	ALLOWED DISRUPTIONS
<code>AGE</code>			
<code>zk-pdb</code>	2	<code>N/A</code>	1
	<code>7s</code>		

The non-zero value for `ALLOWED DISRUPTIONS` means that the disruption controller has seen the pods, counted the matching pods, and updated the status of the PDB.

You can get more information about the status of a PDB with this command:

```
kubectl get poddisruptionbudgets zk-pdb -o yaml
```

```
apiVersion: policy/v1beta1
kind: PodDisruptionBudget
metadata:
  annotations:
    'creationTimestamp': "2020-03-04T04:22:56Z"
    generation: 1
    name: zk-pdb
  status:
    currentHealthy: 3
    desiredHealthy: 2
    disruptionsAllowed: 1
    expectedPods: 3
    observedGeneration: 1
```

## Arbitrary Controllers and Selectors

You can skip this section if you only use PDBs with the built-in application controllers (`Deployment`, `ReplicationController`, `ReplicaSet`, and `StatefulSet`), with the PDB selector matching the controller's selector.

You can use a PDB with pods controlled by another type of controller, by an "operator", or bare pods, but with these restrictions:

- only `.spec.minAvailable` can be used, not `.spec.maxUnavailable`.
- only an integer value can be used with `.spec.minAvailable`, not a percentage.

You can use a selector which selects a subset or superset of the pods belonging to a built-in controller. The eviction API will disallow eviction of any pod covered by multiple PDBs, so most users will want to avoid

overlapping selectors. One reasonable use of overlapping PDBs is when pods are being transitioned from one PDB to another.

## 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](#).

Last modified October 15, 2020 at 12:45 PM PST: [content/en/docs/tasks/run-application/configure-pdb: Acceptable overlapping PDBs \(71cdde785\)](#)  
[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Protecting an Application with a PodDisruptionBudget](#)
- [Identify an Application to Protect](#)
- [Think about how your application reacts to disruptions](#)
  - [Rounding logic when specifying percentages](#)
- [Specifying a PodDisruptionBudget](#)
- [Create the PDB object](#)
- [Check the status of the PDB](#)
- [Arbitrary Controllers and Selectors](#)

## Run Jobs

Run Jobs using parallel processing.

---

[Running Automated Tasks with a CronJob](#)

[Parallel Processing using Expansions](#)

[Coarse Parallel Processing Using a Work Queue](#)

[Fine Parallel Processing Using a Work Queue](#)

## Running Automated Tasks with a CronJob

You can use a [CronJob](#) to run [Jobs](#) on a time-based schedule. These automated jobs run like [Cron](#) tasks on a Linux or UNIX system.

Cron jobs are useful for creating periodic and recurring tasks, like running backups or sending emails. Cron jobs can also schedule individual tasks for

a specific time, such as if you want to schedule a job for a low activity period.

Cron jobs have limitations and idiosyncrasies. For example, in certain circumstances, a single cron job can create multiple jobs. Therefore, jobs should be idempotent.

For more limitations, see [CronJobs](#).

## Before you begin

- You need to have a Kubernetes cluster, and the `kubectl` command-line tool must be configured to communicate with your cluster. If you do not already have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:
  - [Katacoda](#)
  - [Play with Kubernetes](#)

## Creating a Cron Job

Cron jobs require a config file. This example cron job config `.spec` file prints the current time and a hello message every minute:

[application/job/cronjob.yaml](#)



```
apiVersion: batch/v1beta1
kind: CronJob
metadata:
  name: hello
spec:
  schedule: "*/1 * * * *"
  jobTemplate:
    spec:
      template:
        spec:
          containers:
            - name: hello
              image: busybox
              imagePullPolicy: IfNotPresent
              args:
                - /bin/sh
                - -c
                - date; echo Hello from the Kubernetes cluster
  restartPolicy: OnFailure
```

Run the example CronJob by using this command:

```
kubectl create -f https://k8s.io/examples/application/job/cronjob.yaml
```

The output is similar to this:

```
cronjob.batch/hello created
```

After creating the cron job, get its status using this command:

```
kubectl get cronjob hello
```

The output is similar to this:

NAME	SCHEDULE	SUSPEND	ACTIVE	LAST SCHEDULE	AGE
hello	*/1 * * * *	False	0	<none>	10s

As you can see from the results of the command, the cron job has not scheduled or run any jobs yet. Watch for the job to be created in around one minute:

```
kubectl get jobs --watch
```

The output is similar to this:

NAME	COMPLETIONS	DURATION	AGE
hello-4111706356	0/1	0s	0s
hello-4111706356	0/1	0s	0s
hello-4111706356	1/1	5s	5s

Now you've seen one running job scheduled by the "hello" cron job. You can stop watching the job and view the cron job again to see that it scheduled the job:

```
kubectl get cronjob hello
```

The output is similar to this:

NAME	SCHEDULE	SUSPEND	ACTIVE	LAST SCHEDULE	AGE
hello	*/1 * * * *	False	0	50s	75s

You should see that the cron job `hello` successfully scheduled a job at the time specified in `LAST SCHEDULE`. There are currently 0 active jobs, meaning that the job has completed or failed.

Now, find the pods that the last scheduled job created and view the standard output of one of the pods.

**Note:** The job name and pod name are different.

```
# Replace "hello-4111706356" with the job name in your system
pods=$(kubectl get pods --selector=job-name=hello-4111706356 --output=jsonpath={.items[*].metadata.name})
```

Show pod log:

```
kubectl logs $pods
```

The output is similar to this:

```
Fri Feb 22 11:02:09 UTC 2019
Hello from the Kubernetes cluster
```

## Deleting a Cron Job

When you don't need a cron job any more, delete it with `kubectl delete cronjob <cronjob name>`:

```
kubectl delete cronjob hello
```

Deleting the cron job removes all the jobs and pods it created and stops it from creating additional jobs. You can read more about removing jobs in [garbage collection](#).

## Writing a Cron Job Spec

As with all other Kubernetes configs, a cron job needs `apiVersion`, `kind`, and `metadata` fields. For general information about working with config files, see [deploying applications](#), and [using kubectl to manage resources](#) documents.

A cron job config also needs a [.spec section](#).

**Note:** All modifications to a cron job, especially its `.spec`, are applied only to the following runs.

### Schedule

The `.spec.schedule` is a required field of the `.spec`. It takes a [Cron](#) format string, such as `0 * * * *` or `@hourly`, as schedule time of its jobs to be created and executed.

The format also includes extended vixie cron step values. As explained in the [FreeBSD manual](#):

Step values can be used in conjunction with ranges. Following a range with `/<number>` specifies skips of the number's value through the range. For example, `0-23/2` can be used in the hours field to specify command execution every other hour (the alternative in the V7 standard is `0,2,4,6,8,10,12,14,16,18,20,22`). Steps are also permitted after an asterisk, so if you want to say "every two hours", just use `*/2`.

**Note:** A question mark (?) in the schedule has the same meaning as an asterisk \*, that is, it stands for any of available value for a given field.

### Job Template

The `.spec.jobTemplate` is the template for the job, and it is required. It has exactly the same schema as a [Job](#), except that it is nested and does not have

an `apiVersion` or `kind`. For information about writing a job `.spec`, see [Writing a Job Spec](#).

## Starting Deadline

The `.spec.startingDeadlineSeconds` field is optional. It stands for the deadline in seconds for starting the job if it misses its scheduled time for any reason. After the deadline, the cron job does not start the job. Jobs that do not meet their deadline in this way count as failed jobs. If this field is not specified, the jobs have no deadline.

The CronJob controller counts how many missed schedules happen for a cron job. If there are more than 100 missed schedules, the cron job is no longer scheduled. When `.spec.startingDeadlineSeconds` is not set, the CronJob controller counts missed schedules from `status.lastScheduleTime` until now.

For example, one cron job is supposed to run every minute, the `status.lastScheduleTime` of the cronjob is 5:00am, but now it's 7:00am. That means 120 schedules were missed, so the cron job is no longer scheduled.

If the `.spec.startingDeadlineSeconds` field is set (not null), the CronJob controller counts how many missed jobs occurred from the value of `.spec.startingDeadlineSeconds` until now.

For example, if it is set to 200, it counts how many missed schedules occurred in the last 200 seconds. In that case, if there were more than 100 missed schedules in the last 200 seconds, the cron job is no longer scheduled.

## Concurrency Policy

The `.spec.concurrencyPolicy` field is also optional. It specifies how to treat concurrent executions of a job that is created by this cron job. The spec may specify only one of the following concurrency policies:

- **Allow** (default): The cron job allows concurrently running jobs
- **Forbid**: The cron job does not allow concurrent runs; if it is time for a new job run and the previous job run hasn't finished yet, the cron job skips the new job run
- **Replace**: If it is time for a new job run and the previous job run hasn't finished yet, the cron job replaces the currently running job run with a new job run

Note that concurrency policy only applies to the jobs created by the same cron job. If there are multiple cron jobs, their respective jobs are always allowed to run concurrently.

## **Suspend**

The `.spec.suspend` field is also optional. If it is set to `true`, all subsequent executions are suspended. This setting does not apply to already started executions. Defaults to `false`.

**Caution:** Executions that are suspended during their scheduled time count as missed jobs. When `.spec.suspend` changes from `true` to `false` on an existing cron job without a [starting deadline](#), the missed jobs are scheduled immediately.

## **Jobs History Limits**

The `.spec.successfulJobsHistoryLimit` and `.spec.failedJobsHistoryLimit` fields are optional. These fields specify how many completed and failed jobs should be kept. By default, they are set to 3 and 1 respectively. Setting a limit to 0 corresponds to keeping none of the corresponding kind of jobs after they finish.

## **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](#).

Last modified August 07, 2020 at 8:40 PM PST: [Tune links in tasks section \(2/2\) \(92ae1a9cf\)](#)

[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Creating a Cron Job](#)
- [Deleting a Cron Job](#)
- [Writing a Cron Job Spec](#)
  - [Schedule](#)
  - [Job Template](#)
  - [Starting Deadline](#)
  - [Concurrency Policy](#)
  - [Suspend](#)
  - [Jobs History Limits](#)

## **Parallel Processing using Expansions**

This task demonstrates running multiple [Jobs](#) based on a common template. You can use this approach to process batches of work in parallel.

For this example there are only three items: *apple*, *banana*, and *cherry*. The sample Jobs process each item simply by printing a string then pausing.

See [using Jobs in real workloads](#) to learn about how this pattern fits more realistic use cases.

## Before you begin

You should be familiar with the basic, non-parallel, use of [Job](#).

You need to have a Kubernetes cluster, and the `kubectl` command-line tool must be configured to communicate with your cluster. If you do not already have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:

- [Katacoda](#)
- [Play with Kubernetes](#)

For basic templating you need the command-line utility `sed`.

To follow the advanced templating example, you need a working installation of [Python](#), and the `Jinja2` template library for Python.

Once you have Python set up, you can install `Jinja2` by running:

```
pip install --user jinja2
```

## Create Jobs based on a template

First, download the following template of a Job to a file called `job-tmpl.yaml`. Here's what you'll download:

[application/job/job-tmpl.yaml](#)



```
apiVersion: batch/v1
kind: Job
metadata:
  name: process-item-$ITEM
  labels:
    jobgroup: jobexample
spec:
  template:
    metadata:
      name: jobexample
      labels:
        jobgroup: jobexample
    spec:
      containers:
        - name: C
          image: busybox
```

```
  command: ["sh", "-c", "echo Processing item $ITEM && sleep 5"]
  restartPolicy: Never
```

```
# Use curl to download job-tmpl.yaml
curl -L -s -0 https://k8s.io/examples/application/job/job-tmpl.yaml
```

The file you downloaded is not yet a valid Kubernetes [manifest](#). Instead that template is a YAML representation of a Job object with some placeholders that need to be filled in before it can be used. The `$ITEM` syntax is not meaningful to Kubernetes.

## Create manifests from the template

The following shell snippet uses `sed` to replace the string `$ITEM` with the loop variable, writing into a temporary directory named `jobs`. Run this now:

```
# Expand the template into multiple files, one for each item to be processed.
mkdir ./jobs
for i in apple banana cherry
do
  cat job-tmpl.yaml | sed "s/\$ITEM/$i/" > ./jobs/job-$i.yaml
done
```

Check if it worked:

```
ls jobs/
```

The output is similar to this:

```
job-apple.yaml
job-banana.yaml
job-cherry.yaml
```

You could use any type of template language (for example: `Jinja2`; `ERB`), or write a program to generate the Job manifests.

## Create Jobs from the manifests

Next, create all the Jobs with one `kubectl` command:

```
kubectl create -f ./jobs
```

The output is similar to this:

```
job.batch/process-item-apple created
job.batch/process-item-banana created
job.batch/process-item-cherry created
```

Now, check on the jobs:

```
kubectl get jobs -l jobgroup=jobexample
```

The output is similar to this:

NAME	COMPLETIONS	DURATION	AGE
process-item-apple	1/1	14s	22s
process-item-banana	1/1	12s	21s
process-item-cherry	1/1	12s	20s

Using the `-l` option to `kubectl` selects only the Jobs that are part of this group of jobs (there might be other unrelated jobs in the system).

You can check on the Pods as well using the same [label selector](#):

```
kubectl get pods -l jobgroup=jobexample
```

The output is similar to:

NAME	READY	STATUS	RESTARTS	AGE
process-item-apple-kixwv	0/1	Completed	0	4m
process-item-banana-wrsf7	0/1	Completed	0	4m
process-item-cherry-dnfu9	0/1	Completed	0	4m

We can use this single command to check on the output of all jobs at once:

```
kubectl logs -f -l jobgroup=jobexample
```

The output should be:

```
Processing item apple
Processing item banana
Processing item cherry
```

## Clean up

```
# Remove the Jobs you created
# Your cluster automatically cleans up their Pods
kubectl delete job -l jobgroup=jobexample
```

## Use advanced template parameters

In the [first example](#), each instance of the template had one parameter, and that parameter was also used in the Job's name. However, [names](#) are restricted to contain only certain characters.

This slightly more complex example uses the [Jinja template language](#) to generate manifests and then objects from those manifests, with a multiple parameters for each Job.

For this part of the task, you are going to use a one-line Python script to convert the template to a set of manifests.

First, copy and paste the following template of a Job object, into a file called `job.yaml.jinja2`:

```
{%- set params = [{ "name": "apple", "url": "http://dbpedia.org/resource/Apple", },  
                  { "name": "banana", "url": "http://dbpedia.org/resource/Banana", },  
                  { "name": "cherry", "url": "http://dbpedia.org/resource/Cherry" }]  
%}  
{%- for p in params %}  
{%- set name = p["name"] %}  
{%- set url = p["url"] %}  
---  
apiVersion: batch/v1  
kind: Job  
metadata:  
  name: jobexample-{{ name }}  
  labels:  
    jobgroup: jobexample  
spec:  
  template:  
    metadata:  
      name: jobexample  
      labels:  
        jobgroup: jobexample  
    spec:  
      containers:  
      - name: c  
        image: busybox  
        command: ["sh", "-c", "echo Processing URL {{ url }} && sleep 5"]  
      restartPolicy: Never  
{%- endfor %}
```

The above template defines two parameters for each Job object using a list of python dicts (lines 1-4). A `for` loop emits one Job manifest for each set of parameters (remaining lines).

This example relies on a feature of YAML. One YAML file can contain multiple documents (Kubernetes manifests, in this case), separated by `---` on a line by itself. You can pipe the output directly to `kubectl` to create the Jobs.

Next, use this one-line Python program to expand the template:

```
alias render_template='python -c "from jinja2 import Template;  
import sys; print(Template(sys.stdin.read()).render());"'
```

Use `render_template` to convert the parameters and template into a single YAML file containing Kubernetes manifests:

```
# This requires the alias you defined earlier
cat job.yaml.jinja2 | render_template > jobs.yaml
```

You can view `jobs.yaml` to verify that the `render_template` script worked correctly.

Once you are happy that `render_template` is working how you intend, you can pipe its output into `kubectl`:

```
cat job.yaml.jinja2 | render_template | kubectl apply -f -
```

Kubernetes accepts and runs the Jobs you created.

## Clean up

```
# Remove the Jobs you created
# Your cluster automatically cleans up their Pods
kubectl delete job -l jobgroup=jobexample
```

## Using Jobs in real workloads

In a real use case, each Job performs some substantial computation, such as rendering a frame of a movie, or processing a range of rows in a database. If you were rendering a movie you would set `$ITEM` to the frame number. If you were processing rows from a database table, you would set `$ITEM` to represent the range of database rows to process.

In the task, you ran a command to collect the output from Pods by fetching their logs. In a real use case, each Pod for a Job writes its output to durable storage before completing. You can use a PersistentVolume for each Job, or an external storage service. For example, if you are rendering frames for a movie, use HTTP to PUT the rendered frame data to a URL, using a different URL for each frame.

## Labels on Jobs and Pods

After you create a Job, Kubernetes automatically adds additional [labels](#) that distinguish one Job's pods from another Job's pods.

In this example, each Job and its Pod template have a label: `jobgroup=jobexample`.

Kubernetes itself pays no attention to labels named `jobgroup`. Setting a label for all the Jobs you create from a template makes it convenient to operate on all those Jobs at once. In the [first example](#) you used a template to create several Jobs. The template ensures that each Pod also gets the same label, so you can check on all Pods for these templated Jobs with a single command.

**Note:** The label key `jobgroup` is not special or reserved. You can pick your own labelling scheme. There are [recommended labels](#) that you can use if you wish.

## Alternatives

If you plan to create a large number of Job objects, you may find that:

- Even using labels, managing so many Jobs is cumbersome.
- If you create many Jobs in a batch, you might place high load on the Kubernetes control plane. Alternatively, the Kubernetes API server could rate limit you, temporarily rejecting your requests with a 429 status.
- You are limited by a [resource quota](#) on Jobs: the API server permanently rejects some of your requests when you create a great deal of work in one batch.

There are other [job patterns](#) that you can use to process large amounts of work without creating very many Job objects.

You could also consider writing your own [controller](#) to manage Job objects automatically.

## 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](#).

Last modified August 07, 2020 at 8:40 PM PST: [Tune links in tasks section \(2/2\) \(92ae1a9cf\)](#)

[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Create Jobs based on a template](#)
  - [Create manifests from the template](#)
  - [Create Jobs from the manifests](#)
  - [Clean up](#)
- [Use advanced template parameters](#)
  - [Clean up](#)
- [Using Jobs in real workloads](#)
- [Labels on Jobs and Pods](#)
- [Alternatives](#)

# **Coarse Parallel Processing Using a Work Queue**

*In this example, we will run a Kubernetes Job with multiple parallel worker processes.*

*In this example, as each pod is created, it picks up one unit of work from a task queue, completes it, deletes it from the queue, and exits.*

*Here is an overview of the steps in this example:*

1. **Start a message queue service.** In this example, we use RabbitMQ, but you could use another one. In practice you would set up a message queue service once and reuse it for many jobs.
2. **Create a queue, and fill it with messages.** Each message represents one task to be done. In this example, a message is just an integer that we will do a lengthy computation on.
3. **Start a Job that works on tasks from the queue.** The Job starts several pods. Each pod takes one task from the message queue, processes it, and repeats until the end of the queue is reached.

## **Before you begin**

*Be familiar with the basic, non-parallel, use of [Job](#).*

*You need to have a Kubernetes cluster, and the kubectl command-line tool must be configured to communicate with your cluster. If you do not already have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:*

- [Katacoda](#)
- [Play with Kubernetes](#)

## **Starting a message queue service**

*This example uses RabbitMQ, but it should be easy to adapt to another AMQP-type message service.*

*In practice you could set up a message queue service once in a cluster and reuse it for many jobs, as well as for long-running services.*

*Start RabbitMQ as follows:*

```
kubectl create -f https://raw.githubusercontent.com/kubernetes/kubernetes/release-1.3/examples/celery-rabbitmq/rabbitmq-service.yaml
```

```
service "rabbitmq-service" created
```

```
kubectl create -f https://raw.githubusercontent.com/kubernetes/kubernetes/release-1.3/examples/celery-rabbitmq/rabbitmq-controller.yaml
```

```
replicationcontroller "rabbitmq-controller" created
```

We will only use the rabbitmq part from the [celery-rabbitmq example](#).

## Testing the message queue service

Now, we can experiment with accessing the message queue. We will create a temporary interactive pod, install some tools on it, and experiment with queues.

First create a temporary interactive Pod.

```
# Create a temporary interactive container
kubectl run -i --tty temp --image ubuntu:18.04
```

```
Waiting for pod default/temp-loe07 to be running, status is
Pending, pod ready: false
... [ previous line repeats several times .. hit return when it
stops ] ...
```

Note that your pod name and command prompt will be different.

Next install the amqp-tools so we can work with message queues.

```
# Install some tools
root@temp-loe07:/# apt-get update
.... [ lots of output ] ....
root@temp-loe07:/# apt-get install -y curl ca-certificates amqp-
tools python dnsutils
.... [ lots of output ] ....
```

Later, we will make a docker image that includes these packages.

Next, we will check that we can discover the rabbitmq service:

```
# Note the rabbitmq-service has a DNS name, provided by
Kubernetes:
```

```
root@temp-loe07:/# nslookup rabbitmq-service
Server:          10.0.0.10
Address:        10.0.0.10#53
```

```
Name:    rabbitmq-service.default.svc.cluster.local
Address: 10.0.147.152
```

```
# Your address will vary.
```

If Kube-DNS is not setup correctly, the previous step may not work for you. You can also find the service IP in an env var:

```
# env | grep RABBIT | grep HOST
RABBITMQ_SERVICE_SERVICE_HOST=10.0.147.152
# Your address will vary.
```

Next we will verify we can create a queue, and publish and consume messages.

```
# In the next line, rabbitmq-service is the hostname where the
# rabbitmq-service
# can be reached. 5672 is the standard port for rabbitmq.

root@temp-loe07:/# export BROKER_URL=amqp://guest:guest@rabbitmq-
service:5672
# If you could not resolve "rabbitmq-service" in the previous
step,
# then use this command instead:
# root@temp-loe07:/# BROKER_URL=amqp://
guest:guest@$RABBITMQ_SERVICE_SERVICE_HOST:5672

# Now create a queue:

root@temp-loe07:/# /usr/bin/amqp-declare-queue --url=$BROKER_URL
-q foo -d
foo

# Publish one message to it:

root@temp-loe07:/# /usr/bin/amqp-publish --url=$BROKER_URL -r
foo -p -b Hello

# And get it back.

root@temp-loe07:/# /usr/bin/amqp-consume --url=$BROKER_URL -q
foo -c 1 cat && echo
Hello
root@temp-loe07:/#
```

In the last command, the `amqp-consume` tool takes one message (`-c 1`) from the queue, and passes that message to the standard input of an arbitrary command. In this case, the program `cat` is just printing out what it gets on the standard input, and the `echo` is just to add a carriage return so the example is readable.

## Filling the Queue with tasks

Now let's fill the queue with some "tasks". In our example, our tasks are just strings to be printed.

In practice, the content of the messages might be:

- names of files to that need to be processed
- extra flags to the program

- ranges of keys in a database table
- configuration parameters to a simulation
- frame numbers of a scene to be rendered

*In practice, if there is large data that is needed in a read-only mode by all pods of the Job, you will typically put that in a shared file system like NFS and mount that readonly on all the pods, or the program in the pod will natively read data from a cluster file system like HDFS.*

*For our example, we will create the queue and fill it using the amqp command line tools. In practice, you might write a program to fill the queue using an amqp client library.*

```
/usr/bin/amqp-declare-queue --url=$BROKER_URL -q job1 -d job1
```

```
for f in apple banana cherry date fig grape lemon melon
do
    /usr/bin/amqp-publish --url=$BROKER_URL -r job1 -p -b $f
done
```

*So, we filled the queue with 8 messages.*

## Create an Image

*Now we are ready to create an image that we will run as a job.*

*We will use the amqp-consume utility to read the message from the queue and run our actual program. Here is a very simple example program:*

[application/job/rabbitmq/worker.py](#)

```
#!/usr/bin/env python

# Just prints standard out and sleeps for 10 seconds.
import sys
import time
print("Processing " + sys.stdin.readlines()[0])
time.sleep(10)
```

*Give the script execution permission:*

```
chmod +x worker.py
```

*Now, build an image. If you are working in the source tree, then change directory to examples/job/work-queue-1. Otherwise, make a temporary directory, change to it, download the [Dockerfile](#), and [worker.py](#). In either case, build the image with this command:*

```
docker build -t job-wq-1 .
```

For the [Docker Hub](#), tag your app image with your username and push to the Hub with the below commands. Replace <username> with your Hub username.

```
docker tag job-wq-1 <username>/job-wq-1  
docker push <username>/job-wq-1
```

If you are using [Google Container Registry](#), tag your app image with your project ID, and push to GCR. Replace <project> with your project ID.

```
docker tag job-wq-1 gcr.io/<project>/job-wq-1  
gcloud docker -- push gcr.io/<project>/job-wq-1
```

## Defining a Job

Here is a job definition. You'll need to make a copy of the Job and edit the image to match the name you used, and call it `./job.yaml`.

[application/job/rabbitmq/job.yaml](#)  


```
apiVersion: batch/v1  
kind: Job  
metadata:  
  name: job-wq-1  
spec:  
  completions: 8  
  parallelism: 2  
  template:  
    metadata:  
      name: job-wq-1  
    spec:  
      containers:  
      - name: c  
        image: gcr.io/<project>/job-wq-1  
        env:  
        - name: BROKER_URL  
          value: amqp://guest:guest@rabbitmq-service:5672  
        - name: QUEUE  
          value: job1  
  restartPolicy: OnFailure
```

In this example, each pod works on one item from the queue and then exits. So, the completion count of the Job corresponds to the number of work items done. So we set, `.spec.completions: 8` for the example, since we put 8 items in the queue.

## Running the Job

So, now run the Job:

```
kubectl apply -f ./job.yaml
```

Now wait a bit, then check on the job.

```
kubectl describe jobs/job-wq-1
```

```
Name:          job-wq-1
Namespace:     default
Selector:      controller-uid=41d75705-92df-11e7-b85e-
               fa163ee3c11f
Labels:        controller-uid=41d75705-92df-11e7-b85e-
               fa163ee3c11f
Annotations:   <none>
Parallelism:  2
Completions:  8
Start Time:   Wed, 06 Sep 2017 16:42:02 +0800
Pods Statuses: 0 Running / 8 Succeeded / 0 Failed
Pod Template:
  Labels:      controller-uid=41d75705-92df-11e7-b85e-
               fa163ee3c11f
  Containers:
    c:
      Image:      gcr.io/causal-jigsaw-637/job-wq-1
      Port:       8080/TCP
      Environment:
        BROKER_URL: amqp://guest:guest@rabbitmq-service:5672
        QUEUE:       job1
      Mounts:      <none>
      Volumes:     <none>
Events:
FirstSeen  LastSeen  Count  From           SubobjectPath
Type      Reason     Message
27s       27s       1      {job }
Normal    SuccessfulCreate  Created pod: job-wq-1-hcobb
27s       27s       1      {job }
Normal    SuccessfulCreate  Created pod: job-wq-1-weytj
27s       27s       1      {job }
Normal    SuccessfulCreate  Created pod: job-wq-1-qaam5
27s       27s       1      {job }
Normal    SuccessfulCreate  Created pod: job-wq-1-b67sr
26s       26s       1      {job }
Normal    SuccessfulCreate  Created pod: job-wq-1-xe5hj
15s       15s       1      {job }
Normal    SuccessfulCreate  Created pod: job-wq-1-w2zqe
14s       14s       1      {job }
Normal    SuccessfulCreate  Created pod: job-wq-1-d6ppa
```

```
14s      14s      1      {job }
Normal  SuccessfulCreate  Created pod: job-wq-1-p17e0
```

All our pods succeeded. Yay.

## Alternatives

This approach has the advantage that you do not need to modify your "worker" program to be aware that there is a work queue.

It does require that you run a message queue service. If running a queue service is inconvenient, you may want to consider one of the other [job patterns](#).

This approach creates a pod for every work item. If your work items only take a few seconds, though, creating a Pod for every work item may add a lot of overhead. Consider another [example](#), that executes multiple work items per Pod.

In this example, we use the `amqp-consume` utility to read the message from the queue and run our actual program. This has the advantage that you do not need to modify your program to be aware of the queue. A [different example](#), shows how to communicate with the work queue using a client library.

## Caveats

If the number of completions is set to less than the number of items in the queue, then not all items will be processed.

If the number of completions is set to more than the number of items in the queue, then the Job will not appear to be completed, even though all items in the queue have been processed. It will start additional pods which will block waiting for a message.

There is an unlikely race with this pattern. If the container is killed in between the time that the message is acknowledged by the `amqp-consume` command and the time that the container exits with success, or if the node crashes before the kubelet is able to post the success of the pod back to the api-server, then the Job will not appear to be complete, even though all items in the queue have been processed.

## 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](#).

Last modified August 07, 2020 at 8:40 PM PST: [Tune links in tasks section \(2/2\) \(92ae1a9cf\)](#)

[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Starting a message queue service](#)
- [Testing the message queue service](#)
- [Filling the Queue with tasks](#)
- [Create an Image](#)
- [Defining a Job](#)
- [Running the Job](#)
- [Alternatives](#)
- [Caveats](#)

# Fine Parallel Processing Using a Work Queue

In this example, we will run a Kubernetes Job with multiple parallel worker processes in a given pod.

In this example, as each pod is created, it picks up one unit of work from a task queue, processes it, and repeats until the end of the queue is reached.

Here is an overview of the steps in this example:

1. **Start a storage service to hold the work queue.** In this example, we use Redis to store our work items. In the previous example, we used RabbitMQ. In this example, we use Redis and a custom work-queue client library because AMQP does not provide a good way for clients to detect when a finite-length work queue is empty. In practice you would set up a store such as Redis once and reuse it for the work queues of many jobs, and other things.
2. **Create a queue, and fill it with messages.** Each message represents one task to be done. In this example, a message is just an integer that we will do a lengthy computation on.
3. **Start a Job that works on tasks from the queue.** The Job starts several pods. Each pod takes one task from the message queue, processes it, and repeats until the end of the queue is reached.

## Before you begin

You need to have a Kubernetes cluster, and the `kubectl` command-line tool must be configured to communicate with your cluster. If you do not already have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:

- [Katacoda](#)
- [Play with Kubernetes](#)

Be familiar with the basic, non-parallel, use of [Job](#).

# Starting Redis

For this example, for simplicity we will start a single instance of Redis. See the [Redis Example](#) for an example of deploying Redis scalably and redundantly.

You could also download the following files directly:

- [redis-pod.yaml](#)
- [redis-service.yaml](#)
- [Dockerfile](#)
- [job.yaml](#)
- [rediswq.py](#)
- [worker.py](#)

## Filling the Queue with tasks

Now let's fill the queue with some "tasks". In our example, our tasks are just strings to be printed.

Start a temporary interactive pod for running the Redis CLI.

```
kubectl run -i --tty temp --image redis --command "/bin/sh"
Waiting for pod default/redis2-c7h78 to be running, status is
Pending, pod ready: false
Hit enter for command prompt
```

Now hit enter, start the redis CLI, and create a list with some work items in it.

```
# redis-cli -h redis
redis:6379> rpush job2 "apple"
(integer) 1
redis:6379> rpush job2 "banana"
(integer) 2
redis:6379> rpush job2 "cherry"
(integer) 3
redis:6379> rpush job2 "date"
(integer) 4
redis:6379> rpush job2 "fig"
(integer) 5
redis:6379> rpush job2 "grape"
(integer) 6
redis:6379> rpush job2 "lemon"
(integer) 7
redis:6379> rpush job2 "melon"
(integer) 8
redis:6379> rpush job2 "orange"
(integer) 9
redis:6379> lrange job2 0 -1
1) "apple"
```

```
2) "banana"
3) "cherry"
4) "date"
5) "fig"
6) "grape"
7) "lemon"
8) "melon"
9) "orange"
```

So, the list with key `job2` will be our work queue.

Note: if you do not have Kube DNS setup correctly, you may need to change the first step of the above block to `redis-cli -h $REDIS_SERVICE_HOST`.

## Create an Image

Now we are ready to create an image that we will run.

We will use a python worker program with a redis client to read the messages from the message queue.

A simple Redis work queue client library is provided, called `rediswq.py` ([Download](#)).

The "worker" program in each Pod of the Job uses the work queue client library to get work. Here it is:

[application/job/redis/worker.py](#)  


```
#!/usr/bin/env python

import time
import rediswq

host="redis"
# Uncomment next two lines if you do not have Kube-DNS working.
# import os
# host = os.getenv("REDIS_SERVICE_HOST")

q = rediswq.RedisWQ(name="job2", host="redis")
print("Worker with sessionID: " + q.sessionID())
print("Initial queue state: empty=" + str(q.empty()))
while not q.empty():
    item = q.lease(lease_secs=10, block=True, timeout=2)
    if item is not None:
        itemstr = item.decode("utf-8")
        print("Working on " + itemstr)
        time.sleep(10) # Put your actual work here instead of sleep.
        q.complete(item)
    else:
```

```
print("Waiting for work")
print("Queue empty, exiting")
```

You could also download [worker.py](#), [rediswq.py](#), and [Dockerfile](#) files, then build the image:

```
docker build -t job-wq-2 .
```

## Push the image

For the [Docker Hub](#), tag your app image with your username and push to the Hub with the below commands. Replace <username> with your Hub username.

```
docker tag job-wq-2 <username>/job-wq-2
docker push <username>/job-wq-2
```

You need to push to a public repository or [configure your cluster to be able to access your private repository](#).

If you are using [Google Container Registry](#), tag your app image with your project ID, and push to GCR. Replace <project> with your project ID.

```
docker tag job-wq-2 gcr.io/<project>/job-wq-2
gcloud docker -- push gcr.io/<project>/job-wq-2
```

## Defining a Job

Here is the job definition:

[application/job/redis/job.yaml](#)  


```
apiVersion: batch/v1
kind: Job
metadata:
  name: job-wq-2
spec:
  parallelism: 2
  template:
    metadata:
      name: job-wq-2
    spec:
      containers:
        - name: c
          image: gcr.io/myproject/job-wq-2
      restartPolicy: OnFailure
```

Be sure to edit the job template to change `gcr.io/myproject` to your own path.

*In this example, each pod works on several items from the queue and then exits when there are no more items. Since the workers themselves detect when the workqueue is empty, and the Job controller does not know about the workqueue, it relies on the workers to signal when they are done working. The workers signal that the queue is empty by exiting with success. So, as soon as any worker exits with success, the controller knows the work is done, and the Pods will exit soon. So, we set the completion count of the Job to 1. The job controller will wait for the other pods to complete too.*

## Running the Job

*So, now run the Job:*

```
kubectl apply -f ./job.yaml
```

*Now wait a bit, then check on the job.*

```
kubectl describe jobs/job-wq-2
Name:           job-wq-2
Namespace:      default
Selector:       controller-uid=b1c7e4e3-92e1-11e7-b85e-
                fa163ee3c11f
Labels:         controller-uid=b1c7e4e3-92e1-11e7-b85e-
                fa163ee3c11f
Annotations:    <none>
Parallelism:   2
Completions:   <unset>
Start Time:    Mon, 11 Jan 2016 17:07:59 -0800
Pods Statuses: 1 Running / 0 Succeeded / 0 Failed
Pod Template:
  Labels:      controller-uid=b1c7e4e3-92e1-11e7-b85e-
                fa163ee3c11f
                job-name=job-wq-2
  Containers:
    c:
      Image:      gcr.io/exampleproject/job-wq-2
      Port:       8080/TCP
      Environment: <none>
      Mounts:     <none>
      Volumes:    <none>
Events:
FirstSeen  LastSeen  Count  From             SubobjectPath  Type        Reason          Message
-----  -----  -----  -----  -----  -----  -----  -----
33s       33s       1      {job-controller }  Normal       SuccessfulCreate  Created pod: job-wq-2-lglf8
```

```
kubectl logs pods/job-wq-2-7r7b2
Worker with sessionID: bbd72d0a-9e5c-4dd6-abf6-416cc267991f
Initial queue state: empty=False
Working on banana
Working on date
Working on lemon
```

As you can see, one of our pods worked on several work units.

## Alternatives

If running a queue service or modifying your containers to use a work queue is inconvenient, you may want to consider one of the other [job patterns](#).

If you have a continuous stream of background processing work to run, then consider running your background workers with a `ReplicaSet` instead, and consider running a background processing library such as <https://github.com/resque/resque>.

## 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](#).

Last modified August 07, 2020 at 8:40 PM PST: [Tune links in tasks section \(2/2\) \(92ae1a9cf\)](#)

[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Starting Redis](#)
- [Filling the Queue with tasks](#)
- [Create an Image](#)
  - [Push the image](#)
- [Defining a Job](#)
- [Running the Job](#)
- [Alternatives](#)

## Access Applications in a Cluster

Configure load balancing, port forwarding, or setup firewall or DNS configurations to access applications in a cluster.

---

[Web UI \(Dashboard\)](#)

[\*\*Accessing Clusters\*\*](#)

[\*\*Configure Access to Multiple Clusters\*\*](#)

[\*\*Use Port Forwarding to Access Applications in a Cluster\*\*](#)

[\*\*Use a Service to Access an Application in a Cluster\*\*](#)

[\*\*Connect a Front End to a Back End Using a Service\*\*](#)

[\*\*Create an External Load Balancer\*\*](#)

[\*\*List All Container Images Running in a Cluster\*\*](#)

[\*\*Set up Ingress on Minikube with the NGINX Ingress Controller\*\*](#)

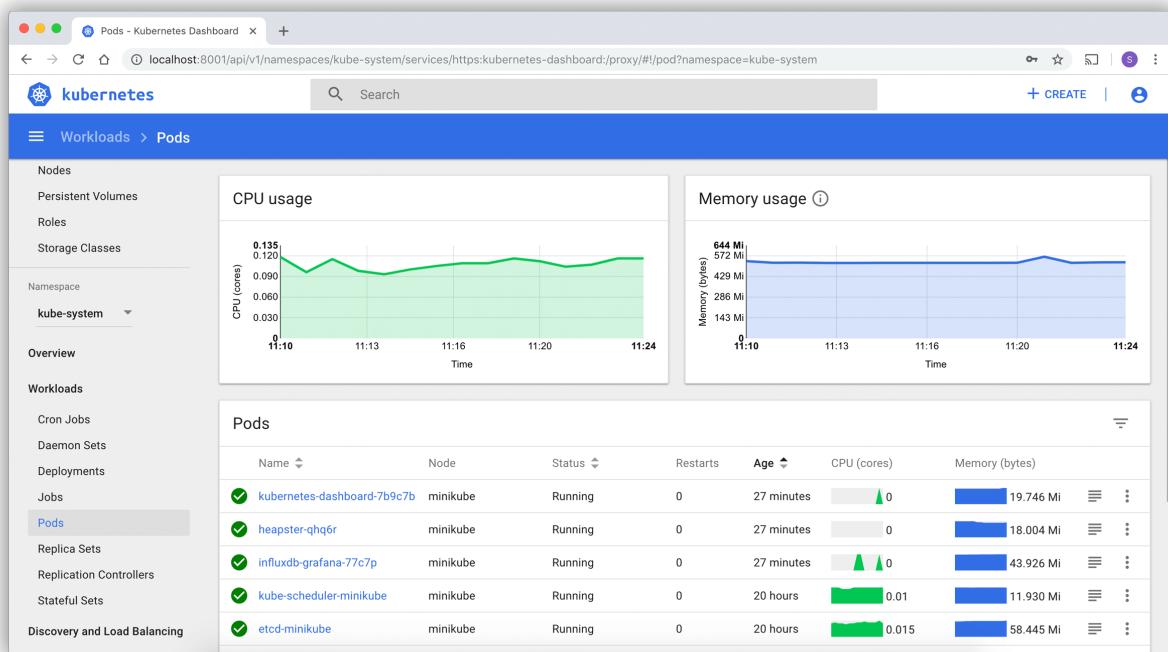
[\*\*Communicate Between Containers in the Same Pod Using a Shared Volume\*\*](#)

[\*\*Configure DNS for a Cluster\*\*](#)

## **Web UI (Dashboard)**

*Dashboard is a web-based Kubernetes user interface. You can use Dashboard to deploy containerized applications to a Kubernetes cluster, troubleshoot your containerized application, and manage the cluster resources. You can use Dashboard to get an overview of applications running on your cluster, as well as for creating or modifying individual Kubernetes resources (such as Deployments, Jobs, DaemonSets, etc). For example, you can scale a Deployment, initiate a rolling update, restart a pod or deploy new applications using a deploy wizard.*

*Dashboard also provides information on the state of Kubernetes resources in your cluster and on any errors that may have occurred.*



## Deploying the Dashboard UI

*The Dashboard UI is not deployed by default. To deploy it, run the following command:*

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes/dashboard/v2.0.0/aio/deploy/recommended.yaml
```

## Accessing the Dashboard UI

*To protect your cluster data, Dashboard deploys with a minimal RBAC configuration by default. Currently, Dashboard only supports logging in with a Bearer Token. To create a token for this demo, you can follow our guide on [creating a sample user](#).*

**Warning:** The sample user created in the tutorial will have administrative privileges and is for educational purposes only.

## Command line proxy

*You can access Dashboard using the kubectl command-line tool by running the following command:*

```
kubectl proxy
```

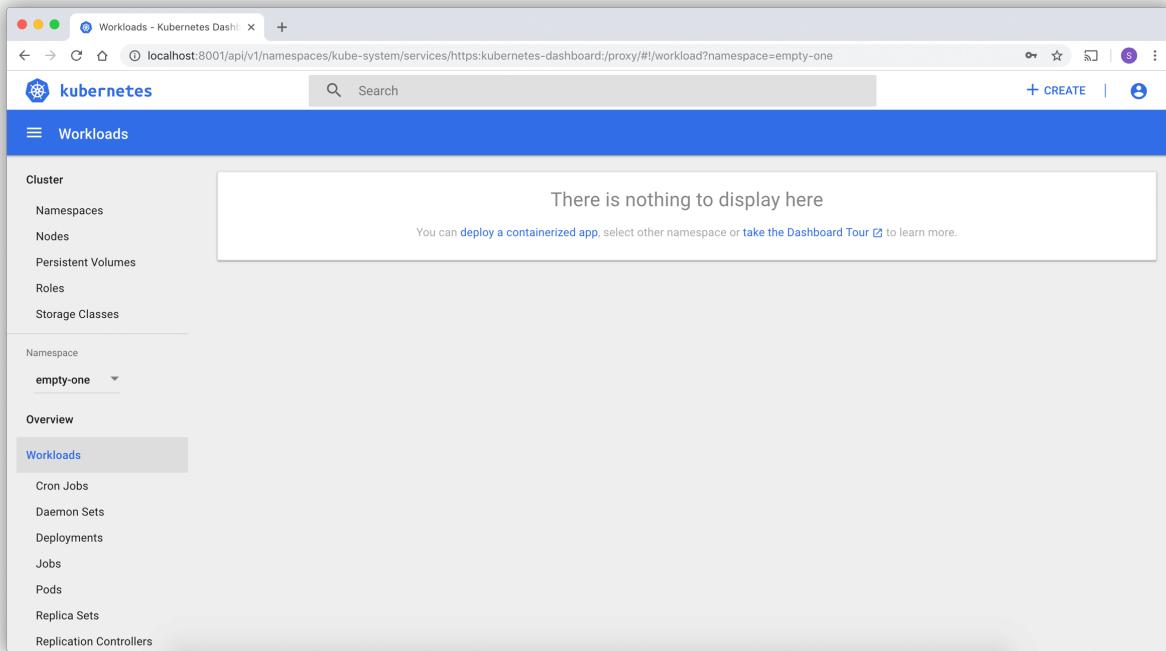
*Kubectl will make Dashboard available at <http://localhost:8001/api/v1/namespaces/kubernetes-dashboard/services/https:kubernetes-dashboard:/proxy/>.*

*The UI can only be accessed from the machine where the command is executed. See `kubectl proxy --help` for more options.*

**Note:** Kubeconfig Authentication method does NOT support external identity providers or x509 certificate-based authentication.

## Welcome view

*When you access Dashboard on an empty cluster, you'll see the welcome page. This page contains a link to this document as well as a button to deploy your first application. In addition, you can view which system applications are running by default in the `kube-system` namespace of your cluster, for example the Dashboard itself.*



## Deploying containerized applications

*Dashboard lets you create and deploy a containerized application as a Deployment and optional Service with a simple wizard. You can either manually specify application details, or upload a YAML or JSON file containing application configuration.*

*Click the **CREATE** button in the upper right corner of any page to begin.*

## **Specifying application details**

The deploy wizard expects that you provide the following information:

- **App name** (mandatory): Name for your application. A [label](#) with the name will be added to the Deployment and Service, if any, that will be deployed.

*The application name must be unique within the selected Kubernetes [namespace](#). It must start with a lowercase character, and end with a lowercase character or a number, and contain only lowercase letters, numbers and dashes (-). It is limited to 24 characters. Leading and trailing spaces are ignored.*

- **Container image** (mandatory): The URL of a public Docker [container image](#) on any registry, or a private image (commonly hosted on the Google Container Registry or Docker Hub). The container image specification must end with a colon.
- **Number of pods** (mandatory): The target number of Pods you want your application to be deployed in. The value must be a positive integer.

*A [Deployment](#) will be created to maintain the desired number of Pods across your cluster.*

- **Service** (optional): For some parts of your application (e.g. frontends) you may want to expose a [Service](#) onto an external, maybe public IP address outside of your cluster (external Service).

**Note:** For external Services, you may need to open up one or more ports to do so.

*Other Services that are only visible from inside the cluster are called internal Services.*

*Irrespective of the Service type, if you choose to create a Service and your container listens on a port (incoming), you need to specify two ports. The Service will be created mapping the port (incoming) to the target port seen by the container. This Service will route to your deployed Pods. Supported protocols are TCP and UDP. The internal DNS name for this Service will be the value you specified as application name above.*

*If needed, you can expand the **Advanced options** section where you can specify more settings:*

- **Description:** The text you enter here will be added as an [annotation](#) to the Deployment and displayed in the application's details.
- **Labels:** Default [labels](#) to be used for your application are application name and version. You can specify additional labels to be applied to the Deployment, Service (if any), and Pods, such as release, environment, tier, partition, and release track.

*Example:*

```
release=1.0
tier=frontend
environment=prod
track=stable
```

- **Namespace:** Kubernetes supports multiple virtual clusters backed by the same physical cluster. These virtual clusters are called [namespaces](#). They let you partition resources into logically named groups.

*Dashboard offers all available namespaces in a dropdown list, and allows you to create a new namespace. The namespace name may contain a maximum of 63 alphanumeric characters and dashes (-) but can not contain capital letters. Namespace names should not consist of only numbers. If the name is set as a number, such as 10, the pod will be put in the default namespace.*

*In case the creation of the namespace is successful, it is selected by default. If the creation fails, the first namespace is selected.*

- **Image Pull Secret:** In case the specified Docker container image is private, it may require [pull secret](#) credentials.

*Dashboard offers all available secrets in a dropdown list, and allows you to create a new secret. The secret name must follow the DNS domain name syntax, for example new.image-pull.secret. The content of a secret must be base64-encoded and specified in a [.docker cfg](#) file. The secret name may consist of a maximum of 253 characters.*

*In case the creation of the image pull secret is successful, it is selected by default. If the creation fails, no secret is applied.*

- **CPU requirement (cores) and Memory requirement (MiB):** You can specify the minimum [resource limits](#) for the container. By default, Pods run with unbounded CPU and memory limits.

- **Run command and Run command arguments:** By default, your containers run the specified Docker image's default [entrypoint command](#). You can use the command options and arguments to override the default.

- **Run as privileged:** This setting determines whether processes in [privileged containers](#) are equivalent to processes running as root on the host. Privileged containers can make use of capabilities like manipulating the network stack and accessing devices.

- **Environment variables:** Kubernetes exposes Services through [environment variables](#). You can compose environment variable or pass arguments to your commands using the values of environment variables. They can be used in applications to find a Service. Values can reference other variables using the \$(VAR\_NAME) syntax.

## **Uploading a YAML or JSON file**

*Kubernetes supports declarative configuration. In this style, all configuration is stored in YAML or JSON configuration files using the Kubernetes [API](#) resource schemas.*

*As an alternative to specifying application details in the deploy wizard, you can define your application in YAML or JSON files, and upload the files using Dashboard.*

## **Using Dashboard**

*Following sections describe views of the Kubernetes Dashboard UI; what they provide and how can they be used.*

### **Navigation**

*When there are Kubernetes objects defined in the cluster, Dashboard shows them in the initial view. By default only objects from the default namespace are shown and this can be changed using the namespace selector located in the navigation menu.*

*Dashboard shows most Kubernetes object kinds and groups them in a few menu categories.*

### **Admin Overview**

*For cluster and namespace administrators, Dashboard lists Nodes, Namespaces and Persistent Volumes and has detail views for them. Node list view contains CPU and memory usage metrics aggregated across all Nodes. The details view shows the metrics for a Node, its specification, status, allocated resources, events and pods running on the node.*

### **Workloads**

*Shows all applications running in the selected namespace. The view lists applications by workload kind (e.g., Deployments, Replica Sets, Stateful Sets, etc.) and each workload kind can be viewed separately. The lists summarize actionable information about the workloads, such as the number of ready pods for a Replica Set or current memory usage for a Pod.*

*Detail views for workloads show status and specification information and surface relationships between objects. For example, Pods that Replica Set is controlling or New Replica Sets and Horizontal Pod Autoscalers for Deployments.*

### **Services**

*Shows Kubernetes resources that allow for exposing services to external world and discovering them within a cluster. For that reason, Service and*

*Ingress views show Pods targeted by them, internal endpoints for cluster connections and external endpoints for external users.*

## **Storage**

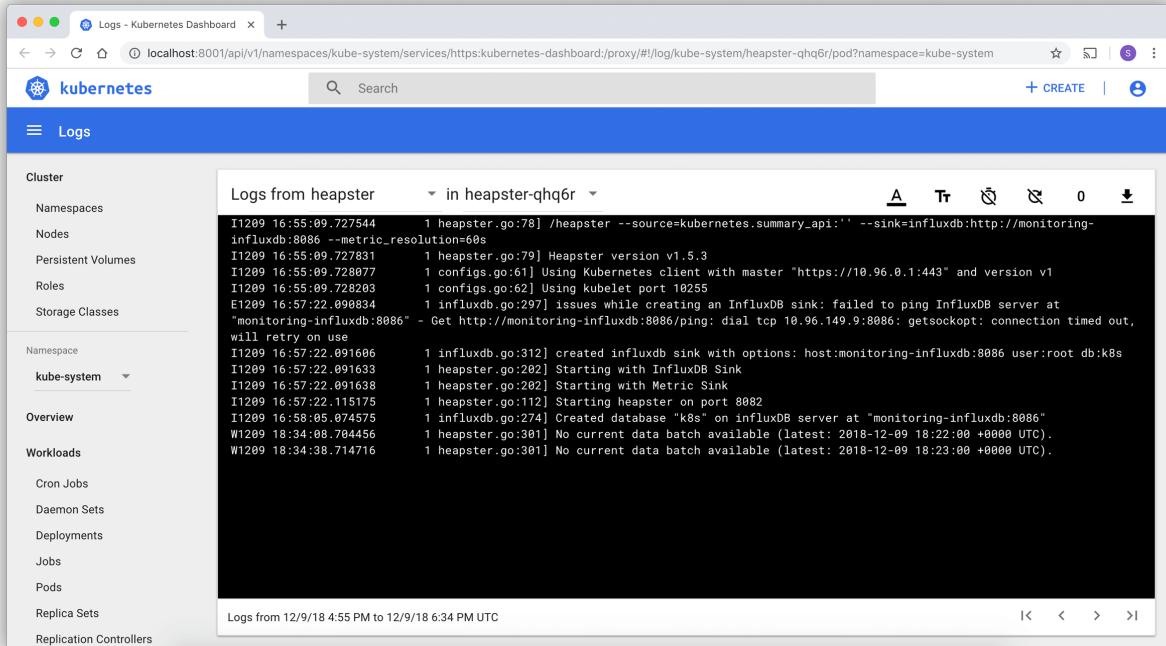
*Storage view shows Persistent Volume Claim resources which are used by applications for storing data.*

## **Config Maps and Secrets**

*Shows all Kubernetes resources that are used for live configuration of applications running in clusters. The view allows for editing and managing config objects and displays secrets hidden by default.*

## **Logs viewer**

*Pod lists and detail pages link to a logs viewer that is built into Dashboard. The viewer allows for drilling down logs from containers belonging to a single Pod.*



## **What's next**

*For more information, see the [Kubernetes Dashboard project page](#).*

## **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](#).

Last modified September 23, 2020 at 6:36 PM PST: [Make the dashboard URL a clickable link \(8c626d69d\)](#)

[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Deploying the Dashboard UI](#)
- [Accessing the Dashboard UI](#)
  - [Command line proxy](#)
- [Welcome view](#)
- [Deploying containerized applications](#)
  - [Specifying application details](#)
  - [Uploading a YAML or JSON file](#)
- [Using Dashboard](#)
  - [Navigation](#)
- [What's next](#)

# Accessing Clusters

This topic discusses multiple ways to interact with clusters.

## Accessing for the first time with kubectl

When accessing the Kubernetes API for the first time, we suggest using the Kubernetes CLI, `kubectl`.

To access a cluster, you need to know the location of the cluster and have credentials to access it. Typically, this is automatically set-up when you work through a [Getting started guide](#), or someone else setup the cluster and provided you with credentials and a location.

Check the location and credentials that `kubectl` knows about with this command:

```
kubectl config view
```

Many of the [examples](#) provide an introduction to using `kubectl` and complete documentation is found in the [kubectl manual](#).

## **Directly accessing the REST API**

Kubectl handles locating and authenticating to the apiserver. If you want to directly access the REST API with an http client like curl or wget, or a browser, there are several ways to locate and authenticate:

- Run kubectl in proxy mode.
  - Recommended approach.
  - Uses stored apiserver location.
  - Verifies identity of apiserver using self-signed cert. No MITM possible.
  - Authenticates to apiserver.
  - In future, may do intelligent client-side load-balancing and failover.
- Provide the location and credentials directly to the http client.
  - Alternate approach.
  - Works with some types of client code that are confused by using a proxy.
  - Need to import a root cert into your browser to protect against MITM.

### **Using kubectl proxy**

The following command runs kubectl in a mode where it acts as a reverse proxy. It handles locating the apiserver and authenticating. Run it like this:

```
kubectl proxy --port=8080
```

See [kubectl proxy](#) for more details.

Then you can explore the API with curl, wget, or a browser, replacing localhost with [::1] for IPv6, like so:

```
curl http://localhost:8080/api/
```

The output is similar to this:

```
{
  "kind": "APIVersions",
  "versions": [
    "v1"
  ],
  "serverAddressByClientCIDRs": [
    {
      "clientCIDR": "0.0.0.0/0",
      "serverAddress": "10.0.1.149:443"
    }
  ]
}
```

## Without kubectl proxy

Use `kubectl describe secret...` to get the token for the default service account with grep/cut:

```
APISERVER=$(kubectl config view --minify | grep server | cut -f 2- -d ":" | tr -d " ")  
SECRET_NAME=$(kubectl get secrets | grep ^default | cut -f1 -d ' ')  
TOKEN=$(kubectl describe secret $SECRET_NAME | grep -E '^token' | cut -f2 -d ':' | tr -d " ")  
  
curl $APISERVER/api --header "Authorization: Bearer $TOKEN" --insecure
```

The output is similar to this:

```
{  
  "kind": "APIVersions",  
  "versions": [  
    "v1"  
,  
  {"  
    "serverAddressByClientCIDRs": [  
      {  
        "clientCIDR": "0.0.0.0/0",  
        "serverAddress": "10.0.1.149:443"  
      }  
    ]  
  }  
}
```

Using jsonpath:

```
APISERVER=$(kubectl config view --minify -o jsonpath='{.clusters[0].cluster.server}')  
SECRET_NAME=$(kubectl get serviceaccount default -o jsonpath='{.secrets[0].name}')  
TOKEN=$(kubectl get secret $SECRET_NAME -o jsonpath='{.data.token}' | base64 --decode)  
  
curl $APISERVER/api --header "Authorization: Bearer $TOKEN" --insecure
```

The output is similar to this:

```
{  
  "kind": "APIVersions",  
  "versions": [  
    "v1"  
,  
  {"  
    "serverAddressByClientCIDRs": [  
      {  
        "clientCIDR": "0.0.0.0/0",  
        "serverAddress": "10.0.1.149:443"  
      }  
    ]  
  }  
}
```

```
    }  
]  
}
```

The above examples use the `--insecure` flag. This leaves it subject to MITM attacks. When `kubectl` accesses the cluster it uses a stored root certificate and client certificates to access the server. (These are installed in the `~/.kube` directory). Since cluster certificates are typically self-signed, it may take special configuration to get your http client to use root certificate.

On some clusters, the apiserver does not require authentication; it may serve on localhost, or be protected by a firewall. There is not a standard for this. [Controlling Access to the API](#) describes how a cluster admin can configure this.

## Programmatic access to the API

Kubernetes officially supports [Go](#) and [Python](#) client libraries.

### Go client

- To get the library, run the following command: `go get k8s.io/client-go@kubernetes-<kubernetes-version-number>`, see [INSTALL.md](#) for detailed installation instructions. See <https://github.com/kubernetes/client-go> to see which versions are supported.
- Write an application atop of the client-go clients. Note that client-go defines its own API objects, so if needed, please import API definitions from client-go rather than from the main repository, e.g., `import "k8s.io/client-go/kubernetes"` is correct.

The Go client can use the same [kubeconfig file](#) as the `kubectl` CLI does to locate and authenticate to the apiserver. See this [example](#).

If the application is deployed as a Pod in the cluster, please refer to the [next section](#).

### Python client

To use [Python client](#), run the following command: `pip install kubernetes`. See [Python Client Library page](#) for more installation options.

The Python client can use the same [kubeconfig file](#) as the `kubectl` CLI does to locate and authenticate to the apiserver. See this [example](#).

### Other languages

There are [client libraries](#) for accessing the API from other languages. See documentation for other libraries for how they authenticate.

## **Accessing the API from a Pod**

*When accessing the API from a pod, locating and authenticating to the apiserver are somewhat different.*

*The recommended way to locate the apiserver within the pod is with the `kubernetes.default.svc` DNS name, which resolves to a Service IP which in turn will be routed to an apiserver.*

*The recommended way to authenticate to the apiserver is with a [service account](#) credential. By kube-system, a pod is associated with a service account, and a credential (token) for that service account is placed into the filesystem tree of each container in that pod, at `/var/run/secrets/kubernetes.io/serviceaccount/token`.*

*If available, a certificate bundle is placed into the filesystem tree of each container at `/var/run/secrets/kubernetes.io/serviceaccount/ca.crt`, and should be used to verify the serving certificate of the apiserver.*

*Finally, the default namespace to be used for namespaced API operations is placed in a file at `/var/run/secrets/kubernetes.io/serviceaccount/namespace` in each container.*

*From within a pod the recommended ways to connect to API are:*

- Run `kubectl proxy` in a sidecar container in the pod, or as a background process within the container. This proxies the Kubernetes API to the localhost interface of the pod, so that other processes in any container of the pod can access it.
- Use the Go client library, and create a client using the `rest.InClusterConfig()` and `kubernetes.NewForConfig()` functions. They handle locating and authenticating to the apiserver. [example](#)

*In each case, the credentials of the pod are used to communicate securely with the apiserver.*

## **Accessing services running on the cluster**

*The previous section was about connecting the Kubernetes API server. This section is about connecting to other services running on Kubernetes cluster. In Kubernetes, the [nodes](#), [pods](#) and [services](#) all have their own IPs. In many cases, the node IPs, pod IPs, and some service IPs on a cluster will not be routable, so they will not be reachable from a machine outside the cluster, such as your desktop machine.*

## **Ways to connect**

You have several options for connecting to nodes, pods and services from outside the cluster:

- Access services through public IPs.
  - Use a service with type *NodePort* or *LoadBalancer* to make the service reachable outside the cluster. See the [services](#) and [kubectl expose](#) documentation.
  - Depending on your cluster environment, this may just expose the service to your corporate network, or it may expose it to the internet. Think about whether the service being exposed is secure. Does it do its own authentication?
  - Place pods behind services. To access one specific pod from a set of replicas, such as for debugging, place a unique label on the pod and create a new service which selects this label.
  - In most cases, it should not be necessary for application developer to directly access nodes via their nodeIPs.
- Access services, nodes, or pods using the Proxy Verb.
  - Does apiserver authentication and authorization prior to accessing the remote service. Use this if the services are not secure enough to expose to the internet, or to gain access to ports on the node IP, or for debugging.
  - Proxies may cause problems for some web applications.
  - Only works for HTTP/HTTPS.
  - Described [here](#).
- Access from a node or pod in the cluster.
  - Run a pod, and then connect to a shell in it using [kubectl exec](#). Connect to other nodes, pods, and services from that shell.
  - Some clusters may allow you to ssh to a node in the cluster. From there you may be able to access cluster services. This is a non-standard method, and will work on some clusters but not others. Browsers and other tools may or may not be installed. Cluster DNS may not work.

## **Discovering builtin services**

Typically, there are several services which are started on a cluster by kube-system. Get a list of these with the `kubectl cluster-info` command:

```
kubectl cluster-info
```

The output is similar to this:

```
Kubernetes master is running at https://104.197.5.247
elasticsearch-logging is running at https://104.197.5.247/api/v1/
namespaces/kube-system/services/elasticsearch-logging/proxy
kibana-logging is running at https://104.197.5.247/api/v1/
namespaces/kube-system/services/kibana-logging/proxy
kube-dns is running at https://104.197.5.247/api/v1/namespaces/
kube-system/services/kube-dns/proxy
grafana is running at https://104.197.5.247/api/v1/namespaces/
```

```
kube-system/services/monitoring-grafana/proxy  
heapster is running at https://104.197.5.247/api/v1/namespaces/  
kube-system/services/monitoring-heapster/proxy
```

This shows the proxy-verb URL for accessing each service. For example, this cluster has cluster-level logging enabled (using Elasticsearch), which can be reached at `https://104.197.5.247/api/v1/namespaces/kube-system/services/elasticsearch-logging/proxy/` if suitable credentials are passed. Logging can also be reached through a kubectl proxy, for example at: `http://localhost:8080/api/v1/namespaces/kube-system/services/elasticsearch-logging/proxy/`. (See [Access Clusters Using the Kubernetes API](#) for how to pass credentials or use kubectl proxy.)

## Manually constructing apiserver proxy URLs

As mentioned above, you use the `kubectl cluster-info` command to retrieve the service's proxy URL. To create proxy URLs that include service endpoints, suffixes, and parameters, you simply append to the service's proxy URL: `http://kubernetes_master_address/api/v1/namespaces/namespace_name/services/service_name[:port_name]/proxy`

If you haven't specified a name for your port, you don't have to specify `port_name` in the URL.

By default, the API server proxies to your service using http. To use https, prefix the service name with https:: `http://kubernetes_master_address/api/v1/namespaces/namespace_name/services/https:service_name:[port_name]/proxy`

The supported formats for the name segment of the URL are:

- `<service_name>` - proxies to the default or unnamed port using http
- `<service_name>:<port_name>` - proxies to the specified port using http
- `https:<service_name>:` - proxies to the default or unnamed port using https (note the trailing colon)
- `https:<service_name>:<port_name>` - proxies to the specified port using https

## Examples

- To access the Elasticsearch service endpoint `_search?q=user:kimchy`, you would use: `http://104.197.5.247/api/v1/namespaces/kube-system/services/elasticsearch-logging/proxy/_search?q=user:kimchy`
- To access the Elasticsearch cluster health information `_cluster/health?pretty=true`, you would use: `https://104.197.5.247/api/v1/namespaces/kube-system/services/elasticsearch-logging/proxy/_cluster/health?pretty=true`

```
{  
  "cluster_name" : "kubernetes_logging",  
  "status" : "yellow",
```

```

"timed_out" : false,
"number_of_nodes" : 1,
"number_of_data_nodes" : 1,
"active_primary_shards" : 5,
"active_shards" : 5,
"relocating_shards" : 0,
"initializing_shards" : 0,
"unassigned_shards" : 5
}

```

## **Using web browsers to access services running on the cluster**

You may be able to put an apiserver proxy url into the address bar of a browser. However:

- Web browsers cannot usually pass tokens, so you may need to use basic (password) auth. Apiserver can be configured to accept basic auth, but your cluster may not be configured to accept basic auth.
- Some web apps may not work, particularly those with client side javascript that construct urls in a way that is unaware of the proxy path prefix.

## **Requesting redirects**

The redirect capabilities have been deprecated and removed. Please use a proxy (see below) instead.

## **So Many Proxies**

There are several different proxies you may encounter when using Kubernetes:

### 1. The [kubectl proxy](#):

- runs on a user's desktop or in a pod
- proxies from a localhost address to the Kubernetes apiserver
- client to proxy uses HTTP
- proxy to apiserver uses HTTPS
- locates apiserver
- adds authentication headers

### 2. The [apiserver proxy](#):

- is a bastion built into the apiserver
- connects a user outside of the cluster to cluster IPs which otherwise might not be reachable
- runs in the apiserver processes
- client to proxy uses HTTPS (or http if apiserver so configured)
- proxy to target may use HTTP or HTTPS as chosen by proxy using available information

- can be used to reach a Node, Pod, or Service
- does load balancing when used to reach a Service

### 3. The [kube proxy](#):

- runs on each node
- proxies UDP and TCP
- does not understand HTTP
- provides load balancing
- is just used to reach services

### 4. A Proxy/Load-balancer in front of apiserver(s):

- existence and implementation varies from cluster to cluster (e.g. nginx)
- sits between all clients and one or more apiservers
- acts as load balancer if there are several apiservers.

### 5. Cloud Load Balancers on external services:

- are provided by some cloud providers (e.g. AWS ELB, Google Cloud Load Balancer)
- are created automatically when the Kubernetes service has type LoadBalancer
- use UDP/TCP only
- implementation varies by cloud provider.

Kubernetes users will typically not need to worry about anything other than the first two types. The cluster admin will typically ensure that the latter types are setup correctly.

## 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](#).

Last modified October 13, 2020 at 12:41 AM PST: [Transfer "Controlling Access to the Kubernetes API" to the Concepts section \(78351ecaf\)](#)  
[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Accessing for the first time with kubectl](#)
- [Directly accessing the REST API](#)
  - [Using kubectl proxy](#)
  - [Without kubectl proxy](#)
- [Programmatic access to the API](#)
  - [Go client](#)
  - [Python client](#)
  - [Other languages](#)

- [Accessing the API from a Pod](#)
- [Accessing services running on the cluster](#)
  - [Ways to connect](#)
  - [Discovering builtin services](#)
  - [Using web browsers to access services running on the cluster](#)
- [Requesting redirects](#)
- [So Many Proxies](#)

# Configure Access to Multiple Clusters

This page shows how to configure access to multiple clusters by using configuration files. After your clusters, users, and contexts are defined in one or more configuration files, you can quickly switch between clusters by using the `kubectl config use-context` command.

**Note:** A file that is used to configure access to a cluster is sometimes called a `kubeconfig` file. This is a generic way of referring to configuration files. It does not mean that there is a file named `kubeconfig`.

## Before you begin

You need to have a Kubernetes cluster, and the `kubectl` command-line tool must be configured to communicate with your cluster. If you do not already have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:

- [Katacoda](#)
- [Play with Kubernetes](#)

To check that `kubectl` is installed, run `kubectl version --client`. The `kubectl` version should be [within one minor version](#) of your cluster's API server.

## Define clusters, users, and contexts

Suppose you have two clusters, one for development work and one for scratch work. In the development cluster, your frontend developers work in a namespace called `frontend`, and your storage developers work in a namespace called `storage`. In your scratch cluster, developers work in the default namespace, or they create auxiliary namespaces as they see fit. Access to the development cluster requires authentication by certificate. Access to the scratch cluster requires authentication by username and password.

Create a directory named `config-exercise`. In your `config-exercise` directory, create a file named `config-demo` with this content:

```

apiVersion: v1
kind: Config
preferences: {}

clusters:
- cluster:
  name: development
- cluster:
  name: scratch

users:
- name: developer
- name: experimenter

contexts:
- context:
  name: dev-frontend
- context:
  name: dev-storage
- context:
  name: exp-scratch

```

A configuration file describes clusters, users, and contexts. Your `config-demo` file has the framework to describe two clusters, two users, and three contexts.

Go to your `config-exercise` directory. Enter these commands to add cluster details to your configuration file:

```

kubectl config --kubeconfig=config-demo set-cluster development
--server=https://1.2.3.4 --certificate-authority=fake-ca-file
kubectl config --kubeconfig=config-demo set-cluster scratch --
server=https://5.6.7.8 --insecure-skip-tls-verify

```

Add user details to your configuration file:

```

kubectl config --kubeconfig=config-demo set-credentials
developer --client-certificate=fake-cert-file --client-key=fake-
key-seefile
kubectl config --kubeconfig=config-demo set-credentials
experimenter --username=exp --password=some-password

```

### **Note:**

- To delete a user you can run `kubectl --kubeconfig=config-demo config unset users.<name>`
- To remove a cluster, you can run `kubectl --kubeconfig=config-demo config unset clusters.<name>`
- To remove a context, you can run `kubectl --kubeconfig=config-demo config unset contexts.<name>`

Add context details to your configuration file:

```
kubectl config --kubeconfig=config-demo set-context dev-frontend  
--cluster=development --namespace=frontend --user=developer  
kubectl config --kubeconfig=config-demo set-context dev-storage  
--cluster=development --namespace=storage --user=developer  
kubectl config --kubeconfig=config-demo set-context exp-scratch  
--cluster=scratch --namespace=default --user=experimenter
```

Open your `config-demo` file to see the added details. As an alternative to opening the `config-demo` file, you can use the `config view` command.

```
kubectl config --kubeconfig=config-demo view
```

The output shows the two clusters, two users, and three contexts:

```
apiVersion: v1  
clusters:  
- cluster:  
    certificate-authority: fake-ca-file  
    server: https://1.2.3.4  
    name: development  
- cluster:  
    insecure-skip-tls-verify: true  
    server: https://5.6.7.8  
    name: scratch  
contexts:  
- context:  
    cluster: development  
    namespace: frontend  
    user: developer  
    name: dev-frontend  
- context:  
    cluster: development  
    namespace: storage  
    user: developer  
    name: dev-storage  
- context:  
    cluster: scratch  
    namespace: default  
    user: experimenter  
    name: exp-scratch  
current-context: ""  
kind: Config  
preferences: {}  
users:  
- name: developer  
  user:  
    client-certificate: fake-cert-file  
    client-key: fake-key-file  
- name: experimenter  
  user:  
    password: some-password  
    username: exp
```

The `fake-ca-file`, `fake-cert-file` and `fake-key-file` above are the placeholders for the pathnames of the certificate files. You need to change these to the actual pathnames of certificate files in your environment.

Sometimes you may want to use Base64-encoded data embedded here instead of separate certificate files; in that case you need add the suffix `-data` to the keys, for example, `certificate-authority-data`, `client-certificate-data`, `client-key-data`.

Each context is a triple (`cluster`, `user`, `namespace`). For example, the `dev-frontend` context says, "Use the credentials of the `developer` user to access the `frontend` namespace of the development cluster".

Set the current context:

```
kubectl config --kubeconfig=config-demo use-context dev-frontend
```

Now whenever you enter a `kubectl` command, the action will apply to the cluster, and namespace listed in the `dev-frontend` context. And the command will use the credentials of the user listed in the `dev-frontend` context.

To see only the configuration information associated with the current context, use the `--minify` flag.

```
kubectl config --kubeconfig=config-demo view --minify
```

The output shows configuration information associated with the `dev-frontend` context:

```
apiVersion: v1
clusters:
- cluster:
    certificate-authority: fake-ca-file
    server: https://1.2.3.4
    name: development
contexts:
- context:
    cluster: development
    namespace: frontend
    user: developer
    name: dev-frontend
current-context: dev-frontend
kind: Config
preferences: {}
users:
- name: developer
  user:
    client-certificate: fake-cert-file
    client-key: fake-key-file
```

Now suppose you want to work for a while in the scratch cluster.

Change the current context to `exp-scratch`:

```
kubectl config --kubeconfig=config-demo use-context exp-scratch
```

Now any `kubectl` command you give will apply to the default namespace of the `scratch` cluster. And the command will use the credentials of the user listed in the `exp-scratch` context.

View configuration associated with the new current context, `exp-scratch`.

```
kubectl config --kubeconfig=config-demo view --minify
```

Finally, suppose you want to work for a while in the `storage` namespace of the development cluster.

Change the current context to `dev-storage`:

```
kubectl config --kubeconfig=config-demo use-context dev-storage
```

View configuration associated with the new current context, `dev-storage`.

```
kubectl config --kubeconfig=config-demo view --minify
```

## Create a second configuration file

In your `config-exercise` directory, create a file named `config-demo-2` with this content:

```
apiVersion: v1
kind: Config
preferences: {}

contexts:
- context:
  cluster: development
  namespace: ramp
  user: developer
  name: dev-ramp-up
```

The preceding configuration file defines a new context named `dev-ramp-up`.

## Set the KUBECONFIG environment variable

See whether you have an environment variable named `KUBECONFIG`. If so, save the current value of your `KUBECONFIG` environment variable, so you can restore it later. For example:

### Linux

```
export KUBECONFIG_SAVED=$KUBECONFIG
```

## **Windows PowerShell**

```
$Env:KUBECONFIG_SAVED=$Env:KUBECONFIG
```

The `KUBECONFIG` environment variable is a list of paths to configuration files. The list is colon-delimited for Linux and Mac, and semicolon-delimited for Windows. If you have a `KUBECONFIG` environment variable, familiarize yourself with the configuration files in the list.

Temporarily append two paths to your `KUBECONFIG` environment variable. For example:

## **Linux**

```
export KUBECONFIG=$KUBECONFIG:config-demo:config-demo-2
```

## **Windows PowerShell**

```
$Env:KUBECONFIG=( "config-demo;config-demo-2" )
```

In your `config-exercise` directory, enter this command:

```
kubectl config view
```

The output shows merged information from all the files listed in your `KUBECONFIG` environment variable. In particular, notice that the merged information has the `dev-ramp-up` context from the `config-demo-2` file and the three contexts from the `config-demo` file:

```
contexts:
- context:
    cluster: development
    namespace: frontend
    user: developer
    name: dev-frontend
- context:
    cluster: development
    namespace: ramp
    user: developer
    name: dev-ramp-up
- context:
    cluster: development
    namespace: storage
    user: developer
    name: dev-storage
- context:
    cluster: scratch
    namespace: default
    user: experimenter
    name: exp-scratch
```

For more information about how kubeconfig files are merged, see [Organizing Cluster Access Using kubeconfig Files](#)

## Explore the \$HOME/.kube directory

If you already have a cluster, and you can use `kubectl` to interact with the cluster, then you probably have a file named `config` in the `$HOME/.kube` directory.

Go to `$HOME/.kube`, and see what files are there. Typically, there is a file named `config`. There might also be other configuration files in this directory. Briefly familiarize yourself with the contents of these files.

## Append \$HOME/.kube/config to your KUBECONFIG environment variable

If you have a `$HOME/.kube/config` file, and it's not already listed in your `KUBECOMFIG` environment variable, append it to your `KUBECONFIG` environment variable now. For example:

### Linux

```
export KUBECONFIG=$KUBECONFIG:$HOME/.kube/config
```

### Windows Powershell

```
$Env:KUBECONFIG="$Env:KUBECONFIG;$HOME\.kube\config"
```

View configuration information merged from all the files that are now listed in your `KUBECONFIG` environment variable. In your config-exercise directory, enter:

```
kubectl config view
```

## Clean up

Return your `KUBECONFIG` environment variable to its original value. For example:

### Linux

```
export KUBECONFIG=$KUBECONFIG_SAVED
```

### Windows PowerShell

```
$Env:KUBECONFIG=$Env:KUBECONFIG_SAVED
```

## What's next

- [Organizing Cluster Access Using kubeconfig Files](#)
- [kubectl config](#)

## 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](#).

Last modified October 22, 2020 at 2:28 PM PST: [Update configure-access-multiple-clusters.md \(d8912d5da\)](#)

[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Define clusters, users, and contexts](#)
- [Create a second configuration file](#)
- [Set the KUBECONFIG environment variable](#)
  - [Linux](#)
  - [Windows PowerShell](#)
  - [Linux](#)
  - [Windows PowerShell](#)
- [Explore the \\$HOME/.kube directory](#)
- [Append \\$HOME/.kube/config to your KUBECONFIG environment variable](#)
  - [Linux](#)
  - [Windows Powershell](#)
- [Clean up](#)
  - [Linux](#)
  - [Windows PowerShell](#)
- [What's next](#)

## Use Port Forwarding to Access Applications in a Cluster

This page shows how to use `kubectl port-forward` to connect to a Redis server running in a Kubernetes cluster. This type of connection can be useful for database debugging.

## Before you begin

- You need to have a Kubernetes cluster, and the `kubectl` command-line tool must be configured to communicate with your cluster. If you do not

*already have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:*

- [Katacoda](#)
- [Play with Kubernetes](#)

*Your Kubernetes server must be at or later than version v1.10. To check the version, enter `kubectl version`.*

- *Install [redis-cli](#).*

## ***Creating Redis deployment and service***

*1. Create a Deployment that runs Redis:*

```
kubectl apply -f https://k8s.io/examples/application/guestbook/redis-master-deployment.yaml
```

*The output of a successful command verifies that the deployment was created:*

```
deployment.apps/redis-master created
```

*View the pod status to check that it is ready:*

```
kubectl get pods
```

*The output displays the pod created:*

NAME	READY	STATUS
RESTARTS AGE		
redis-master-765d459796-258hz 0 50s	1/1	Running

*View the Deployment's status:*

```
kubectl get deployment
```

*The output displays that the Deployment was created:*

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
redis-master	1/1	1	1	55s

*The Deployment automatically manages a ReplicaSet. View the ReplicaSet status using:*

```
kubectl get replicaset
```

*The output displays that the ReplicaSet was created:*

NAME	DESIRED	CURRENT	READY	AGE
redis-master-765d459796	1	1	1	1m

*2. Create a Service to expose Redis on the network:*

```
kubectl apply -f https://k8s.io/examples/application/guestbook/redis-master-service.yaml
```

The output of a successful command verifies that the Service was created:

```
service/redis-master created
```

Check the Service created:

```
kubectl get service redis-master
```

The output displays the service created:

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP
PORT(S)	AGE		
redis-master	ClusterIP	10.0.0.213	<none>
TCP 27s			6379/

- Verify that the Redis server is running in the Pod, and listening on port 6379:

```
# Change redis-master-765d459796-258hz to the name of the Pod
kubectl get pod redis-master-765d459796-258hz --template='{{(index .spec.containers 0).ports 0).containerPort}}\n{{"\\n"}}'
```

The output displays the port for Redis in that Pod:

```
6379
```

(this is the TCP port allocated to Redis on the internet).

## **Forward a local port to a port on the Pod**

1. `kubectl port-forward` allows using resource name, such as a pod name, to select a matching pod to port forward to.

```
# Change redis-master-765d459796-258hz to the name of the Pod
kubectl port-forward redis-master-765d459796-258hz 7000:6379
```

which is the same as

```
kubectl port-forward pods/redis-master-765d459796-258hz
7000:6379
```

or

```
kubectl port-forward deployment/redis-master 7000:6379
```

or

```
kubectl port-forward replicaset/redis-master 7000:6379
```

or

```
kubectl port-forward service/redis-master 7000:redis
```

Any of the above commands works. The output is similar to this:

```
Forwarding from 127.0.0.1:7000 -> 6379
Forwarding from [::1]:7000 -> 6379
```

**Note:** `kubectl port-forward` does not return. To continue with the exercises, you will need to open another terminal.

1. Start the Redis command line interface:

```
redis-cli -p 7000
```

2. At the Redis command line prompt, enter the `ping` command:

```
ping
```

A successful ping request returns:

```
PONG
```

## Optionally let `kubectl` choose the local port

If you don't need a specific local port, you can let `kubectl` choose and allocate the local port and thus relieve you from having to manage local port conflicts, with the slightly simpler syntax:

```
kubectl port-forward deployment/redis-master :6379
```

The `kubectl` tool finds a local port number that is not in use (avoiding low ports numbers, because these might be used by other applications). The output is similar to:

```
Forwarding from 127.0.0.1:62162 -> 6379
Forwarding from [::1]:62162 -> 6379
```

## Discussion

Connections made to local port 7000 are forwarded to port 6379 of the Pod that is running the Redis server. With this connection in place, you can use your local workstation to debug the database that is running in the Pod.

**Note:** `kubectl port-forward` is implemented for TCP ports only. The support for UDP protocol is tracked in [issue 47862](#).

## What's next

Learn more about [kubectl port-forward](#).

# 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](#).

Last modified December 15, 2020 at 11:54 AM PST: [add note to kubectl port-forward \(aec2737f3\)](#)

[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Creating Redis deployment and service](#)
- [Forward a local port to a port on the Pod](#)
  - [Optionally let kubectl choose the local port](#)
- [Discussion](#)
- [What's next](#)

# Use a Service to Access an Application in a Cluster

This page shows how to create a Kubernetes Service object that external clients can use to access an application running in a cluster. The Service provides load balancing for an application that has two running instances.

## Before you begin

You need to have a Kubernetes cluster, and the `kubectl` command-line tool must be configured to communicate with your cluster. If you do not already have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:

- [Katacoda](#)
- [Play with Kubernetes](#)

To check the version, enter `kubectl version`.

## Objectives

- Run two instances of a Hello World application.
- Create a Service object that exposes a node port.
- Use the Service object to access the running application.

# **Creating a service for an application running in two pods**

Here is the configuration file for the application Deployment:

[service/access/hello-application.yaml](#)  


```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hello-world
spec:
  selector:
    matchLabels:
      run: load-balancer-example
  replicas: 2
  template:
    metadata:
      labels:
        run: load-balancer-example
    spec:
      containers:
        - name: hello-world
          image: gcr.io/google-samples/node-hello:1.0
          ports:
            - containerPort: 8080
              protocol: TCP
```

1. Run a Hello World application in your cluster: Create the application Deployment using the file above:

```
kubectl apply -f https://k8s.io/examples/service/access/hello-application.yaml
```

The preceding command creates a [Deployment](#) and an associated [ReplicaSet](#). The ReplicaSet has two [Pods](#) each of which runs the Hello World application.

2. Display information about the Deployment:

```
kubectl get deployments hello-world
kubectl describe deployments hello-world
```

3. Display information about your ReplicaSet objects:

```
kubectl get replicsets
kubectl describe replicsets
```

4. Create a Service object that exposes the deployment:

```
kubectl expose deployment hello-world --type=NodePort --name=example-service
```

5. Display information about the Service:

```
kubectl describe services example-service
```

The output is similar to this:

Name:	example-service
Namespace:	default
Labels:	run=load-balancer-example
Annotations:	<none>
Selector:	run=load-balancer-example
Type:	NodePort
IP:	10.32.0.16
Port:	<unset> 8080/TCP
TargetPort:	8080/TCP
NodePort:	<unset> 31496/TCP
Endpoints:	10.200.1.4:8080,10.200.2.5:8080
Session Affinity:	None
Events:	<none>

Make a note of the NodePort value for the service. For example, in the preceding output, the NodePort value is 31496.

6. List the pods that are running the Hello World application:

```
kubectl get pods --selector="run=load-balancer-example" --output=wide
```

The output is similar to this:

NAME	READY	STATUS	...
IP NODE			
hello-world-2895499144-bsbk5 10.200.1.4 worker1	1/1	Running	...
hello-world-2895499144-m1pwt 10.200.2.5 worker2	1/1	Running	...

7. Get the public IP address of one of your nodes that is running a Hello World pod. How you get this address depends on how you set up your cluster. For example, if you are using Minikube, you can see the node address by running `kubectl cluster-info`. If you are using Google Compute Engine instances, you can use the `gcloud compute instances list` command to see the public addresses of your nodes.

8. On your chosen node, create a firewall rule that allows TCP traffic on your node port. For example, if your Service has a NodePort value of 31568, create a firewall rule that allows TCP traffic on port 31568. Different cloud providers offer different ways of configuring firewall rules.

Use the node address and node port to access the Hello World application:

```
curl http://<public-node-ip>:<node-port>
```

where `<public-node-ip>` is the public IP address of your node, and `<node-port>` is the NodePort value for your service. The response to a successful request is a hello message:

```
Hello Kubernetes!
```

## Using a service configuration file

As an alternative to using `kubectl expose`, you can use a [service configuration file](#) to create a Service.

## Cleaning up

To delete the Service, enter this command:

```
kubectl delete services example-service
```

To delete the Deployment, the ReplicaSet, and the Pods that are running the Hello World application, enter this command:

```
kubectl delete deployment hello-world
```

## What's next

Learn more about [connecting applications with services](#).

## 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](#).

Last modified July 27, 2020 at 4:18 AM PST: [Revise Pod concept \(#22603\)](#)  
[\(49eee8fd3\)](#)

[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Objectives](#)
- [Creating a service for an application running in two pods](#)
- [Using a service configuration file](#)
- [Cleaning up](#)
- [What's next](#)

# Connect a Front End to a Back End Using a Service

This task shows how to create a frontend and a backend microservice. The backend microservice is a hello greeter. The frontend and backend are connected using a Kubernetes [Service](#) object.

## Objectives

- Create and run a microservice using a [Deployment](#) object.
- Route traffic to the backend using a frontend.
- Use a [Service](#) object to connect the frontend application to the backend application.

## Before you begin

You need to have a Kubernetes cluster, and the `kubectl` command-line tool must be configured to communicate with your cluster. If you do not already have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:

- [Katacoda](#)
- [Play with Kubernetes](#)

To check the version, enter `kubectl version`.

This task uses [Services with external load balancers](#), which require a supported environment. If your environment does not support this, you can use a [Service](#) of type [NodePort](#) instead.

## Creating the backend using a Deployment

The backend is a simple hello greeter microservice. Here is the configuration file for the backend Deployment:

[service/access/hello.yaml](#)



```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hello
spec:
  selector:
    matchLabels:
      app: hello
      tier: backend
      track: stable
  replicas: 7
  template:
    metadata:
      labels:
        app: hello
        tier: backend
        track: stable
    spec:
      containers:
        - name: hello
          image: "gcr.io/google-samples/hello-go-gke:1.0"
          ports:
            - name: http
              containerPort: 80
```

Create the backend Deployment:

```
kubectl apply -f https://k8s.io/examples/service/access/
hello.yaml
```

View information about the backend Deployment:

```
kubectl describe deployment hello
```

The output is similar to this:

Name:	hello
Namespace:	default
CreationTimestamp:	Mon, 24 Oct 2016 14:21:02 -0700
Labels:	app=hello tier=backend track=stable
Annotations:	deployment.kubernetes.io/ revision=1

```

Selector:
app=hello,tier=backend,track=stable
Replicas: 7 desired | 7 updated | 7 total
| 7 available | 0 unavailable
StrategyType: RollingUpdate
MinReadySeconds: 0
RollingUpdateStrategy: 1 max unavailable, 1 max surge
Pod Template:
  Labels: app=hello
           tier=backend
           track=stable
Containers:
  hello:
    Image:      "gcr.io/google-samples/hello-go-gke:1.0"
    Port:       80/TCP
    Environment: <none>
    Mounts:     <none>
    Volumes:    <none>
Conditions:
  Type        Status  Reason
  ----        ----- 
  Available   True    MinimumReplicasAvailable
  Progressing True    NewReplicaSetAvailable
OldReplicaSets: <none>
NewReplicaSet:  hello-3621623197 (7/7 replicas created)
Events:
...

```

## ***Creating the backend Service object***

*The key to connecting a frontend to a backend is the backend Service. A Service creates a persistent IP address and DNS name entry so that the backend microservice can always be reached. A Service uses [selectors](#) to find the Pods that it routes traffic to.*

*First, explore the Service configuration file:*

[service/access/hello-service.yaml](#)  
□

```

apiVersion: v1
kind: Service
metadata:
  name: hello
spec:
  selector:
    app: hello

```

```
tier: backend
ports:
- protocol: TCP
  port: 80
  targetPort: http
```

In the configuration file, you can see that the Service routes traffic to Pods that have the labels `app: hello` and `tier: backend`.

Create the `hello` Service:

```
kubectl apply -f https://k8s.io/examples/service/access/hello-service.yaml
```

At this point, you have a backend Deployment running, and you have a Service that can route traffic to it.

## Creating the frontend

Now that you have your backend, you can create a frontend that connects to the backend. The frontend connects to the backend worker Pods by using the DNS name given to the backend Service. The DNS name is "hello", which is the value of the `name` field in the preceding Service configuration file.

The Pods in the frontend Deployment run an `nginx` image that is configured to find the `hello` backend Service. Here is the `nginx` configuration file:

[service/access/frontend.conf](#)



```
upstream hello {
    server hello;
}

server {
    listen 80;

    location / {
        proxy_pass http://hello;
    }
}
```

Similar to the backend, the frontend has a Deployment and a Service. The configuration for the Service has type: LoadBalancer, which means that the Service uses the default load balancer of your cloud provider.

[service/access/frontend.yaml](#)



```
apiVersion: v1
kind: Service
metadata:
  name: frontend
spec:
  selector:
    app: hello
    tier: frontend
  ports:
  - protocol: "TCP"
    port: 80
    targetPort: 80
  type: LoadBalancer

---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: frontend
spec:
  selector:
    matchLabels:
      app: hello
      tier: frontend
      track: stable
  replicas: 1
  template:
    metadata:
      labels:
        app: hello
        tier: frontend
        track: stable
    spec:
      containers:
      - name: nginx
        image: "gcr.io/google-samples/hello-frontend:1.0"
        lifecycle:
          preStop:
            exec:
              command: ["/usr/sbin/nginx", "-s", "quit"]
```

Create the frontend Deployment and Service:

```
kubectl apply -f https://k8s.io/examples/service/access/ frontend.yaml
```

The output verifies that both resources were created:

```
deployment.apps/frontend created  
service/frontend created
```

**Note:** The nginx configuration is baked into the [container image](#). A better way to do this would be to use a [ConfigMap](#), so that you can change the configuration more easily.

## Interact with the frontend Service

Once you've created a Service of type `LoadBalancer`, you can use this command to find the external IP:

```
kubectl get service frontend --watch
```

This displays the configuration for the `frontend` Service and watches for changes. Initially, the external IP is listed as `<pending>`:

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)
AGE				
frontend	LoadBalancer	10.51.252.116	<pending>	80/TCP
10s				

As soon as an external IP is provisioned, however, the configuration updates to include the new IP under the `EXTERNAL-IP` heading:

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)
PORT(S)	AGE			
frontend	LoadBalancer	10.51.252.116	XXX.XXX.XXX.XXX	80/TCP
	1m			

That IP can now be used to interact with the `frontend` service from outside the cluster.

## Send traffic through the frontend

The frontend and backends are now connected. You can hit the endpoint by using the curl command on the external IP of your frontend Service.

```
curl http://${EXTERNAL_IP} # replace this with the EXTERNAL-IP  
you saw earlier
```

The output shows the message generated by the backend:

```
{"message": "Hello"}
```

## Cleaning up

To delete the Services, enter this command:

```
kubectl delete services frontend hello
```

To delete the Deployments, the ReplicaSets and the Pods that are running the backend and frontend applications, enter this command:

```
kubectl delete deployment frontend hello
```

## What's next

- Learn more about [Services](#)
- Learn more about [ConfigMaps](#)

## 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](#).

Last modified August 05, 2020 at 3:17 AM PST: [Replace special quote characters with normal ones. \(c6a96128c\)](#)

[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Objectives](#)
- [Before you begin](#)

- [Creating the backend using a Deployment](#)
- [Creating the backend Service object](#)
- [Creating the frontend](#)
- [Interact with the frontend Service](#)
- [Send traffic through the frontend](#)
- [Cleaning up](#)
- [What's next](#)

# Create an External Load Balancer

This page shows how to create an External Load Balancer.

**Note:** This feature is only available for cloud providers or environments which support external load balancers.

When creating a service, you have the option of automatically creating a cloud network load balancer. This provides an externally-accessible IP address that sends traffic to the correct port on your cluster nodes provided your cluster runs in a supported environment and is configured with the correct cloud load balancer provider package.

For information on provisioning and using an Ingress resource that can give services externally-reachable URLs, load balance the traffic, terminate SSL etc., please check the [Ingress](#) documentation.

## Before you begin

- You need to have a Kubernetes cluster, and the `kubectl` command-line tool must be configured to communicate with your cluster. If you do not already have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:
  - [Katacoda](#)
  - [Play with Kubernetes](#)

To check the version, enter `kubectl version`.

## Configuration file

To create an external load balancer, add the following line to your [service configuration file](#):

**type:** `LoadBalancer`

Your configuration file might look like:

```
apiVersion: v1
kind: Service
metadata:
  name: example-service
```

```
spec:  
  selector:  
    app: example  
  ports:  
    - port: 8765  
      targetPort: 9376  
  type: LoadBalancer
```

## Using kubectl

You can alternatively create the service with the `kubectl expose` command and its `--type=LoadBalancer` flag:

```
kubectl expose rc example --port=8765 --target-port=9376 \  
  --name=example-service --type=LoadBalancer
```

This command creates a new service using the same selectors as the referenced resource (in the case of the example above, a replication controller named `example`).

For more information, including optional flags, refer to the [kubectl expose reference](#).

## Finding your IP address

You can find the IP address created for your service by getting the service information through `kubectl`:

```
kubectl describe services example-service
```

which should produce output like this:

Name:	example-service
Namespace:	default
Labels:	<none>
Annotations:	<none>
Selector:	app=example
Type:	LoadBalancer
IP:	10.67.252.103
LoadBalancer Ingress:	192.0.2.89
Port:	<unnamed> 80/TCP
NodePort:	<unnamed> 32445/TCP
Endpoints:	
10.64.0.4:80, 10.64.1.5:80, 10.64.2.4:80	
Session Affinity:	None
Events:	<none>

The IP address is listed next to `LoadBalancer Ingress`.

**Note:** If you are running your service on Minikube, you can find the assigned IP address and port with:

```
minikube service example-service --url
```

## Preserving the client source IP

Due to the implementation of this feature, the source IP seen in the target container is not the original source IP of the client. To enable preservation of the client IP, the following fields can be configured in the service spec (supported in GCE/Google Kubernetes Engine environments):

- `service.spec.externalTrafficPolicy` - denotes if this Service desires to route external traffic to node-local or cluster-wide endpoints. There are two available options: *Cluster* (default) and *Local*. *Cluster* obscures the client source IP and may cause a second hop to another node, but should have good overall load-spreading. *Local* preserves the client source IP and avoids a second hop for LoadBalancer and NodePort type services, but risks potentially imbalanced traffic spreading.
- `service.spec.healthCheckNodePort` - specifies the health check node port (numeric port number) for the service. If `healthCheckNodePort` isn't specified, the service controller allocates a port from your cluster's NodePort range. You can configure that range by setting an API server command line option, `--service-node-port-range`. It will use the user-specified `healthCheckNodePort` value if specified by the client. It only has an effect when `type` is set to `LoadBalancer` and `externalTrafficPolicy` is set to `Local`.

Setting `externalTrafficPolicy` to `Local` in the Service configuration file activates this feature.

```
apiVersion: v1
kind: Service
metadata:
  name: example-service
spec:
  selector:
    app: example
  ports:
    - port: 8765
      targetPort: 9376
  externalTrafficPolicy: Local
  type: LoadBalancer
```

## Garbage Collecting Load Balancers

**FEATURE STATE:** Kubernetes v1.17 [stable]

In usual case, the correlating load balancer resources in cloud provider should be cleaned up soon after a LoadBalancer type Service is deleted. But it is known that there are various corner cases where cloud resources are orphaned after the associated Service is deleted. Finalizer Protection for Service LoadBalancers was introduced to prevent this from happening. By

*using finalizers, a Service resource will never be deleted until the correlating load balancer resources are also deleted.*

*Specifically, if a Service has type LoadBalancer, the service controller will attach a finalizer named `service.kubernetes.io/load-balancer-cleanup`. The finalizer will only be removed after the load balancer resource is cleaned up. This prevents dangling load balancer resources even in corner cases such as the service controller crashing.*

## **External Load Balancer Providers**

*It is important to note that the datapath for this functionality is provided by a load balancer external to the Kubernetes cluster.*

*When the Service type is set to LoadBalancer, Kubernetes provides functionality equivalent to type equals ClusterIP to pods within the cluster and extends it by programming the (external to Kubernetes) load balancer with entries for the Kubernetes pods. The Kubernetes service controller automates the creation of the external load balancer, health checks (if needed), firewall rules (if needed) and retrieves the external IP allocated by the cloud provider and populates it in the service object.*

## **Caveats and Limitations when preserving source IPs**

*GCE/AWS load balancers do not provide weights for their target pools. This was not an issue with the old LB kube-proxy rules which would correctly balance across all endpoints.*

*With the new functionality, the external traffic is not equally load balanced across pods, but rather equally balanced at the node level (because GCE/AWS and other external LB implementations do not have the ability for specifying the weight per node, they balance equally across all target nodes, disregarding the number of pods on each node).*

*We can, however, state that for `NumServicePods << NumNodes` or `NumServicePods >> NumNodes`, a fairly close-to-equal distribution will be seen, even without weights.*

*Once the external load balancers provide weights, this functionality can be added to the LB programming path. Future Work: No support for weights is provided for the 1.4 release, but may be added at a future date*

*Internal pod to pod traffic should behave similar to ClusterIP services, with equal probability across all pods.*

## **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](#).

Last modified May 30, 2020 at 3:10 PM PST: [add en pages \(ecc27bbbe\)](#)  
[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Configuration file](#)
- [Using kubectl](#)
- [Finding your IP address](#)
- [Preserving the client source IP](#)
- [Garbage Collecting Load Balancers](#)
- [External Load Balancer Providers](#)
- [Caveats and Limitations when preserving source IPs](#)

# List All Container Images Running in a Cluster

This page shows how to use `kubectl` to list all of the Container images for Pods running in a cluster.

## Before you begin

You need to have a Kubernetes cluster, and the `kubectl` command-line tool must be configured to communicate with your cluster. If you do not already have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:

- [Katacoda](#)
- [Play with Kubernetes](#)

To check the version, enter `kubectl version`.

In this exercise you will use `kubectl` to fetch all of the Pods running in a cluster, and format the output to pull out the list of Containers for each.

## List all Container images in all namespaces

- Fetch all Pods in all namespaces using `kubectl get pods --all-namespaces`

- Format the output to include only the list of Container image names using `-o jsonpath={..image}`. This will recursively parse out the `image` field from the returned json.
  - See the [jsonpath reference](#) for further information on how to use jsonpath.
- Format the output using standard tools: `tr`, `sort`, `uniq`
  - Use `tr` to replace spaces with newlines
  - Use `sort` to sort the results
  - Use `uniq` to aggregate image counts

```
kubectl get pods --all-namespaces -o jsonpath="{..image}" | \
tr -s '[:space:]' '\n' | \
sort | \
uniq -c
```

The above command will recursively return all fields named `image` for all items returned.

As an alternative, it is possible to use the absolute path to the `image` field within the Pod. This ensures the correct field is retrieved even when the field name is repeated, e.g. many fields are called `name` within a given item:

```
kubectl get pods --all-namespaces -o jsonpath=".items[*].spec.co\
ntainers[*].image"
```

The jsonpath is interpreted as follows:

- `.items[*]`: for each returned value
- `.spec`: get the spec
- `.containers[*]`: for each container
- `.image`: get the image

**Note:** When fetching a single Pod by name, for example `kubectl get pod nginx`, the `.items[*]` portion of the path should be omitted because a single Pod is returned instead of a list of items.

## List Container images by Pod

The formatting can be controlled further by using the `range` operation to iterate over elements individually.

```
kubectl get pods --all-namespaces -o=jsonpath='{range .items[*]}{\
"\n"}{.metadata.name}{":\t}{range .spec.containers[*]}{.image}'
```

```
{", "}{end}{end}' |\\  
sort
```

## ***List Container images filtering by Pod label***

To target only Pods matching a specific label, use the `-l` flag. The following matches only Pods with labels matching `app=nginx`.

```
kubectl get pods --all-namespaces -o=jsonpath=".image" -l  
app=nginx
```

## ***List Container images filtering by Pod namespace***

To target only pods in a specific namespace, use the `namespace` flag. The following matches only Pods in the `kube-system` namespace.

```
kubectl get pods --namespace kube-system -o jsonpath=".image"
```

## ***List Container images using a go-template instead of jsonpath***

As an alternative to jsonpath, Kubectl supports using [go-templates](#) for formatting the output:

```
kubectl get pods --all-namespaces -o go-template --template="{{range .items}}{{range .spec.containers}}{{.image}} {{end}}{{end}}"
```

## **What's next**

### **Reference**

- [Jsonpath](#) reference guide
- [Go template](#) reference guide

# 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](#).

Last modified August 07, 2020 at 8:40 PM PST: [Tune links in tasks section \(2/2\) \(92ae1a9cf\)](#)

[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [List all Container images in all namespaces](#)
- [List Container images by Pod](#)
- [List Container images filtering by Pod label](#)
- [List Container images filtering by Pod namespace](#)
- [List Container images using a go-template instead of jsonpath](#)
- [What's next](#)
  - [Reference](#)

# Set up Ingress on Minikube with the NGINX Ingress Controller

An [Ingress](#) is an API object that defines rules which allow external access to services in a cluster. An [Ingress controller](#) fulfills the rules set in the Ingress.

This page shows you how to set up a simple Ingress which routes requests to Service web or web2 depending on the HTTP URI.

## Before you begin

You need to have a Kubernetes cluster, and the `kubectl` command-line tool must be configured to communicate with your cluster. If you do not already have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:

- [Katacoda](#)
- [Play with Kubernetes](#)

To check the version, enter `kubectl version`.

## Create a Minikube cluster

1. Click **Launch Terminal**

Last modified October 12, 2020 at 11:17 AM PST: [Move example-ingress.yaml \(9fa03cbdf\)](#)

[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Create a Minikube cluster](#)
- [Enable the Ingress controller](#)
- [Deploy a hello, world app](#)
- [Create an Ingress resource](#)
- [Create Second Deployment](#)
- [Edit Ingress](#)
- [Test Your Ingress](#)
- [What's next](#)

# Communicate Between Containers in the Same Pod Using a Shared Volume

This page shows how to use a Volume to communicate between two Containers running in the same Pod. See also how to allow processes to communicate by [sharing process namespace](#) between containers.

## Before you begin

You need to have a Kubernetes cluster, and the kubectl command-line tool must be configured to communicate with your cluster. If you do not already have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:

- [Katacoda](#)
- [Play with Kubernetes](#)

To check the version, enter `kubectl version`.

## Creating a Pod that runs two Containers

In this exercise, you create a Pod that runs two Containers. The two containers share a Volume that they can use to communicate. Here is the configuration file for the Pod:

[pods/two-container-pod.yaml](#)



```
apiVersion: v1
kind: Pod
metadata:
  name: two-containers
spec:
```

```
restartPolicy: Never

volumes:
- name: shared-data
  emptyDir: {}

containers:
- name: nginx-container
  image: nginx
  volumeMounts:
    - name: shared-data
      mountPath: /usr/share/nginx/html

- name: debian-container
  image: debian
  volumeMounts:
    - name: shared-data
      mountPath: /pod-data
    command: ["/bin/sh"]
    args: ["-c", "echo Hello from the debian container > /pod-data/index.html"]
```

*In the configuration file, you can see that the Pod has a Volume named shared-data.*

*The first container listed in the configuration file runs an nginx server. The mount path for the shared Volume is /usr/share/nginx/html. The second container is based on the debian image, and has a mount path of /pod-data. The second container runs the following command and then terminates.*

```
echo Hello from the debian container > /pod-data/index.html
```

*Notice that the second container writes the index.html file in the root directory of the nginx server.*

*Create the Pod and the two Containers:*

```
kubectl apply -f https://k8s.io/examples/pods/two-container-pod.yaml
```

*View information about the Pod and the Containers:*

```
kubectl get pod two-containers --output=yaml
```

*Here is a portion of the output:*

```
apiVersion: v1
kind: Pod
metadata:
  ...
  name: two-containers
  namespace: default
  ...
spec:
  ...
  containerStatuses:
    - containerID: docker://c1d8abd1 ...
      image: debian
      ...
      lastState:
        terminated:
          ...
          name: debian-container
          ...
    - containerID: docker://96c1ff2c5bb ...
      image: nginx
      ...
      name: nginx-container
      ...
      state:
        running:
        ...

```

*You can see that the debian Container has terminated, and the nginx Container is still running.*

*Get a shell to nginx Container:*

```
kubectl exec -it two-containers -c nginx-container -- /bin/
bash
```

*In your shell, verify that nginx is running:*

```
root@two-containers:/# apt-get update
root@two-containers:/# apt-get install curl procps
root@two-containers:/# ps aux
```

*The output is similar to this:*

```
USER      PID ... STAT START    TIME COMMAND
root      1 ... Ss   21:12    0:00 nginx: master process
nginx -g daemon off;
```

Recall that the `debian` Container created the `index.html` file in the `nginx` root directory. Use `curl` to send a GET request to the `nginx` server:

```
root@two-containers:/# curl localhost
```

The output shows that `nginx` serves a web page written by the `debian` container:

```
Hello from the debian container
```

## Discussion

The primary reason that Pods can have multiple containers is to support helper applications that assist a primary application. Typical examples of helper applications are data pullers, data pushers, and proxies. Helper and primary applications often need to communicate with each other. Typically this is done through a shared filesystem, as shown in this exercise, or through the loopback network interface, `localhost`. An example of this pattern is a web server along with a helper program that polls a Git repository for new updates.

The Volume in this exercise provides a way for Containers to communicate during the life of the Pod. If the Pod is deleted and recreated, any data stored in the shared Volume is lost.

## What's next

- Learn more about [patterns for composite containers](#).
- Learn about [composite containers for modular architecture](#).
- See [Configuring a Pod to Use a Volume for Storage](#).
- See [Configure a Pod to share process namespace between containers in a Pod](#)
- See [Volume](#).

See [Pod](#).

o

## 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](#).

Last modified August 07, 2020 at 8:40 PM PST: [Tune links in tasks section \(2/2\) \(92ae1a9cf\)](#)

[Edit this page](#) [Create child page](#) [Create an issue](#)

- o [Before you begin](#)
- o [Creating a Pod that runs two Containers](#)
- o [Discussion](#)
- o [What's next](#)

## Configure DNS for a Cluster

Kubernetes offers a *DNS cluster addon*, which most of the supported environments enable by default. In Kubernetes version 1.11 and later, CoreDNS is recommended and is installed by default with kubeadm.

For more information on how to configure CoreDNS for a Kubernetes cluster, see the [Customizing DNS Service](#). An example demonstrating how to use Kubernetes DNS with kube-dns, see the [Kubernetes DNS sample plugin](#).

## 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](#).

Last modified May 30, 2020 at 3:10 PM PST: [add en pages \(ecc27bbbe\)](#)  
[Edit this page](#) [Create child page](#) [Create an issue](#)

# **Monitoring, Logging, and Debugging**

*Set up monitoring and logging to troubleshoot a cluster, or debug a containerized application.*

---

[\*\*Application Introspection and Debugging\*\*](#)

[\*\*Auditing\*\*](#)

[\*\*Debug a StatefulSet\*\*](#)

[\*\*Debug Init Containers\*\*](#)

[\*\*Debug Pods and ReplicationControllers\*\*](#)

[\*\*Debug Running Pods\*\*](#)

[\*\*Debug Services\*\*](#)

[\*\*Debugging Kubernetes nodes with cubectl\*\*](#)

[\*\*Determine the Reason for Pod Failure\*\*](#)

[\*\*Developing and debugging services locally\*\*](#)

[\*\*Events in Stackdriver\*\*](#)

[\*\*Get a Shell to a Running Container\*\*](#)

[\*\*Logging Using Elasticsearch and Kibana\*\*](#)

[\*\*Logging Using Stackdriver\*\*](#)

[\*\*Monitor Node Health\*\*](#)

[\*\*Resource metrics pipeline\*\*](#)

[\*\*Tools for Monitoring Resources\*\*](#)

[\*\*Troubleshoot Applications\*\*](#)

[\*\*Troubleshoot Clusters\*\*](#)

[\*\*Troubleshooting\*\*](#)

# **Application Introspection and Debugging**

*Once your application is running, you'll inevitably need to debug problems with it. Earlier we described how you can use `kubectl get pods` to retrieve simple status information about your pods. But there are a number of ways to get even more information about your application.*

## **Using `kubectl describe pod` to fetch details about pods**

*For this example we'll use a Deployment to create two pods, similar to the earlier example.*

[application/nginx-with-request.yaml](#)



```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 2
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx
      resources:
        limits:
          memory: "128Mi"
          cpu: "500m"
      ports:
        - containerPort: 80
```

*Create deployment by running following command:*

```
kubectl apply -f https://k8s.io/examples/application/nginx-with-request.yaml
```

```
deployment.apps/nginx-deployment created
```

Check pod status by following command:

```
kubectl get pods
```

NAME	READY	STATUS
RESTARTS	AGE	
nginx-deployment-1006230814-6winp 0	1/1	Running 11s
nginx-deployment-1006230814-fmgu3 0	1/1	Running 11s

We can retrieve a lot more information about each of these pods using `kubectl describe pod`. For example:

```
kubectl describe pod nginx-deployment-1006230814-6winp
```

```
Name:           nginx-deployment-1006230814-6winp
Namespace:      default
Node:           kubernetes-node-wul5/10.240.0.9
Start Time:     Thu, 24 Mar 2016 01:39:49 +0000
Labels:          app=nginx,pod-template-hash=1006230814
Annotations:    kubernetes.io/created-
by={"kind":"SerializedReference","apiVersion":"v1","reference": {"kind":"ReplicaSet","namespace":"default","name":"nginx-deployment-1956810328","uid":"14e607e7-8ba1-11e7-b5cb-fa16"} ...
Status:         Running
IP:             10.244.0.6
Controllers:    ReplicaSet/nginx-deployment-1006230814
Containers:
  nginx:
    Container ID:    docker://90315cc9f513c724e9957a4788d3e625a078de84750f244a40f97ae355eb1149
    Image:           nginx
    Image ID:        docker://6f62f48c4e55d700cf3eb1b5e33fa051802986b77b874cc351cce539e5163707
    Port:            80/TCP
    QoS Tier:
      cpu:           Guaranteed
      memory:        Guaranteed
    Limits:
      cpu:           500m
      memory:        128Mi
    Requests:
      memory:        128Mi
      cpu:           500m
    State:
      Started:       Thu, 24 Mar 2016 01:39:51 +0000
      Ready:          True
      Restart Count:  0
```

*Here you can see configuration information about the container(s) and Pod (labels, resource requirements, etc.), as well as status information about the container(s) and Pod (state, readiness, restart count, events, etc.).*

*The container state is one of Waiting, Running, or Terminated. Depending on the state, additional information will be provided -- here you can see that for a container in Running state, the system tells you when the container started.*

*Ready* tells you whether the container passed its last readiness probe. (In this case, the container does not have a readiness probe configured; the container is assumed to be ready if no readiness probe is configured.)

*Restart Count* tells you how many times the container has been restarted; this information can be useful for detecting crash loops in containers that are configured with a restart policy of 'always.'

Currently the only Condition associated with a Pod is the binary *Ready* condition, which indicates that the pod is able to service requests and should be added to the load balancing pools of all matching services.

Lastly, you see a log of recent events related to your Pod. The system compresses multiple identical events by indicating the first and last time it was seen and the number of times it was seen. "From" indicates the component that is logging the event, "SubobjectPath" tells you which object (e.g. container within the pod) is being referred to, and "Reason" and "Message" tell you what happened.

## Example: debugging Pending Pods

A common scenario that you can detect using events is when you've created a Pod that won't fit on any node. For example, the Pod might request more resources than are free on any node, or it might specify a label selector that doesn't match any nodes. Let's say we created the previous Deployment with 5 replicas (instead of 2) and requesting 600 millicores instead of 500, on a four-node cluster where each (virtual) machine has 1 CPU. In that case one of the Pods will not be able to schedule. (Note that because of the cluster addon pods such as fluentd, skydns, etc., that run on each node, if we requested 1000 millicores then none of the Pods would be able to schedule.)

```
kubectl get pods
```

NAME	READY	STATUS
RESTARTS	AGE	
nginx-deployment-1006230814-6winp	1/1	Running
0 7m		
nginx-deployment-1006230814-fmgu3	1/1	Running
0 7m		
nginx-deployment-1370807587-6ekbw	1/1	Running
0 1m		
nginx-deployment-1370807587-fg172	0/1	Pending
0 1m		
nginx-deployment-1370807587-fz9sd	0/1	Pending
0 1m		

To find out why the nginx-deployment-1370807587-fz9sd pod is not running, we can use `kubectl describe pod` on the pending Pod and look at its events:

```
kubectl describe pod nginx-deployment-1370807587-fz9sd
```

```
Name:           nginx-deployment-1370807587-fz9sd
Namespace:      default
Node:           /
Labels:          app=nginx, pod-template-
hash=1370807587
Status:          Pending
IP:
Controllers:    ReplicaSet/nginx-deployment-1370807587
Containers:
  nginx:
    Image:        nginx
    Port:         80/TCP
    QoS Tier:
      memory:   Guaranteed
      cpu:       Guaranteed
    Limits:
      cpu:       1
      memory:   128Mi
    Requests:
      cpu:       1
      memory:   128Mi
    Environment Variables:
Volumes:
  default-token-4bcbi:
    Type:        Secret (a volume populated by a Secret)
    SecretName:  default-token-4bcbi
Events:
  FirstSeen     LastSeen      Count
From          SubobjectPath
Type          Reason        Message
-----  -----
-----  -----
-----  -----
  1m          48s          7      {default-
scheduler }                               Warning
FailedScheduling      pod (nginx-deployment-1370807587-
fz9sd) failed to fit in any node
  fit failure on node (kubernetes-node-6ta5): Node didn't
have enough resource: CPU, requested: 1000, used: 1420,
capacity: 2000
  fit failure on node (kubernetes-node-wul5): Node didn't
have enough resource: CPU, requested: 1000, used: 1100,
capacity: 2000
```

Here you can see the event generated by the scheduler saying that the Pod failed to schedule for reason FailedScheduling (and possibly others). The message tells us that there were not enough resources for the Pod on any of the nodes.

To correct this situation, you can use `kubectl scale` to update your Deployment to specify four or fewer replicas. (Or you could just leave the one Pod pending, which is harmless.)

Events such as the ones you saw at the end of `kubectl describe pod` are persisted in etcd and provide high-level information on what is happening in the cluster. To list all events you can use

```
kubectl get events
```

but you have to remember that events are namespaced. This means that if you're interested in events for some namespaced object (e.g. what happened with Pods in namespace `my-namespace`) you need to explicitly provide a namespace to the command:

```
kubectl get events --namespace=my-namespace
```

To see events from all namespaces, you can use the `--all-namespaces` argument.

In addition to `kubectl describe pod`, another way to get extra information about a pod (beyond what is provided by `kubectl get pod`) is to pass the `-o yaml` output format flag to `kubectl get pod`. This will give you, in YAML format, even more information than `kubectl describe pod`-essentially all of the information the system has about the Pod. Here you will see things like annotations (which are key-value metadata without the label restrictions, that is used internally by Kubernetes system components), restart policy, ports, and volumes.

```
kubectl get pod nginx-deployment-1006230814-6winp -o yaml
```

```
apiVersion: v1
kind: Pod
metadata:
  annotations:
    kubernetes.io/created-by: /
  {"kind": "SerializedReference", "apiVersion": "v1", "reference": {"kind": "ReplicaSet", "namespace": "default", "name": "nginx-deployment-1006230814", "uid": "4c84c175-f161-11e5-9a78-42010af00005", "apiVersion": "extensions", "resourceVersion": "133434"}}
  creationTimestamp: 2016-03-24T01:39:50Z
  generateName: nginx-deployment-1006230814-
  labels:
    app: nginx
    pod-template-hash: "1006230814"
  name: nginx-deployment-1006230814-6winp
  namespace: default
  resourceVersion: "133447"
  uid: 4c879808-f161-11e5-9a78-42010af00005
spec:
  containers:
```

```
- image: nginx
imagePullPolicy: Always
name: nginx
ports:
- containerPort: 80
  protocol: TCP
resources:
  limits:
    cpu: 500m
    memory: 128Mi
  requests:
    cpu: 500m
    memory: 128Mi
terminationMessagePath: /dev/termination-log
volumeMounts:
- mountPath: /var/run/secrets/kubernetes.io/
serviceaccount
  name: default-token-4bcbi
  readOnly: true
dnsPolicy: ClusterFirst
nodeName: kubernetes-node-wul5
restartPolicy: Always
securityContext: {}
serviceAccount: default
serviceAccountName: default
terminationGracePeriodSeconds: 30
volumes:
- name: default-token-4bcbi
  secret:
    secretName: default-token-4bcbi
status:
  conditions:
- lastProbeTime: null
  lastTransitionTime: 2016-03-24T01:39:51Z
  status: "True"
  type: Ready
  containerStatuses:
- containerID: docker://
90315cc9f513c724e9957a4788d3e625a078de84750f244a40f97ae355eb1
149
  image: nginx
  imageID: docker://
6f62f48c4e55d700cf3eb1b5e33fa051802986b77b874cc351cce539e5163
707
  lastState: {}
  name: nginx
  ready: true
  restartCount: 0
  state:
    running:
      startedAt: 2016-03-24T01:39:51Z
hostIP: 10.240.0.9
```

```

phase: Running
podIP: 10.244.0.6
startTime: 2016-03-24T01:39:49Z

```

## Example: debugging a down/unreachable node

Sometimes when debugging it can be useful to look at the status of a node -- for example, because you've noticed strange behavior of a Pod that's running on the node, or to find out why a Pod won't schedule onto the node. As with Pods, you can use `kubectl describe node` and `kubectl get node -o yaml` to retrieve detailed information about nodes. For example, here's what you'll see if a node is down (disconnected from the network, or kubelet dies and won't restart, etc.). Notice the events that show the node is `NotReady`, and also notice that the pods are no longer running (they are evicted after five minutes of `NotReady` status).

```
kubectl get nodes
```

NAME	STATUS	ROLES	AGE
<i>VERSION</i>			
<i>kubernetes-node-861h</i>	<i>NotReady</i>	<i>&lt;none&gt;</i>	<i>1h</i>
<i>v1.13.0</i>			
<i>kubernetes-node-bols</i>	<i>Ready</i>	<i>&lt;none&gt;</i>	<i>1h</i>
<i>v1.13.0</i>			
<i>kubernetes-node-st6x</i>	<i>Ready</i>	<i>&lt;none&gt;</i>	<i>1h</i>
<i>v1.13.0</i>			
<i>kubernetes-node-unaj</i>	<i>Ready</i>	<i>&lt;none&gt;</i>	<i>1h</i>
<i>v1.13.0</i>			

```
kubectl describe node kubernetes-node-861h
```

<i>Name:</i>	<i>kubernetes-node-861h</i>	
<i>Role</i>		
<i>Labels:</i>	<i>kubernetes.io/arch=amd64</i> <i>kubernetes.io/os=linux</i> <i>kubernetes.io/hostname=kubernetes-node-861h</i>	
<i>Annotations:</i>	<i>node.alpha.kubernetes.io/ttl=0</i> <i>volumes.kubernetes.io/controller-managed-</i> <i>attach-detach=true</i>	
<i>Taints:</i>	<i>&lt;none&gt;</i>	
<i>CreationTimestamp:</i>	<i>Mon, 04 Sep 2017 17:13:23 +0800</i>	
<i>Phase:</i>		
<i>Conditions:</i>		
	<i>Type</i>	<i>Status</i>
<i>LastHeartbeatTime</i>		
<i>LastTransitionTime</i>		
<i>Reason</i>	<i>Message</i>	
-----	-----	-----
-----	-----	-----
<i>OutOfDisk</i>	<i>Unknown</i>	<i>Fri, 08 Sep 2017</i>

```

16:04:28 +0800           Fri, 08 Sep 2017 16:20:58
+0800          NodeStatusUnknown      Kubelet stopped
posting node status.

    MemoryPressure      Unknown      Fri, 08 Sep 2017
16:04:28 +0800           Fri, 08 Sep 2017 16:20:58
+0800          NodeStatusUnknown      Kubelet stopped
posting node status.

    DiskPressure      Unknown      Fri, 08 Sep 2017
16:04:28 +0800           Fri, 08 Sep 2017 16:20:58
+0800          NodeStatusUnknown      Kubelet stopped
posting node status.

    Ready            Unknown      Fri, 08 Sep 2017
16:04:28 +0800           Fri, 08 Sep 2017 16:20:58
+0800          NodeStatusUnknown      Kubelet stopped
posting node status.

Addresses:      10.240.115.55,104.197.0.26
Capacity:
cpu:            2
hugePages:      0
memory:         4046788Ki
pods:           110
Allocatable:
cpu:            1500m
hugePages:      0
memory:         1479263Ki
pods:           110
System Info:
Machine ID:    8e025a21a4254e11b028584d9d8b12c4
System UUID:    E886850C47E3
Boot ID:        5cd18b37-c5bd-4658-94e0-
e436d3f110e0
Kernel Version: 4.4.0-31-generic
OS Image:       Debian GNU/Linux 8 (jessie)
Operating System: linux
Architecture:   amd64
Container Runtime Version: docker://1.12.5
Kubelet Version: v1.6.9+a3d1dfa6f4335
Kube-Proxy Version: v1.6.9+a3d1dfa6f4335
ExternalID:     15233045891481496305
Non-terminated Pods: (9 in total)

    Namespace
Name                                     CPU
Requests    CPU Limits      Memory Requests  Memory Limits
-----  -----
-----  -----
-----  -----
.....
Allocated resources:
    (Total limits may be over 100 percent, i.e.,
overcommitted.)

```

<i>CPU Requests</i>	<i>CPU Limits</i>	<i>Memory Requests</i>
<i>Memory Limits</i>		
-----	-----	-----
900m (60%)	2200m (146%)	1009286400 (66%)
5681286400 (375%)		
<i>Events:</i>	<none>	

```
kubectl get node kubernetes-node-861h -o yaml
```

```
apiVersion: v1
kind: Node
metadata:
  creationTimestamp: 2015-07-10T21:32:29Z
  labels:
    kubernetes.io/hostname: kubernetes-node-861h
  name: kubernetes-node-861h
  resourceVersion: "757"
  uid: 2a69374e-274b-11e5-a234-42010af0d969
spec:
  externalID: "15233045891481496305"
  podCIDR: 10.244.0.0/24
  providerID: gce://striped-torus-760/us-central1-b/
kubernetes-node-861h
status:
  addresses:
  - address: 10.240.115.55
    type: InternalIP
  - address: 104.197.0.26
    type: ExternalIP
  capacity:
    cpu: "1"
    memory: 3800808Ki
    pods: "100"
  conditions:
  - lastHeartbeatTime: 2015-07-10T21:34:32Z
    lastTransitionTime: 2015-07-10T21:35:15Z
    reason: Kubelet stopped posting node status.
    status: Unknown
    type: Ready
  nodeInfo:
    bootID: 4e316776-b40d-4f78-a4ea-ab0d73390897
    containerRuntimeVersion: docker://Unknown
    kernelVersion: 3.16.0-0.bpo.4-amd64
    kubeProxyVersion: v0.21.1-185-gffc5a86098dc01
    kubeletVersion: v0.21.1-185-gffc5a86098dc01
    machineID: ""
    osImage: Debian GNU/Linux 7 (wheezy)
    systemUUID: ABE5F6B4-D44B-108B-C46A-24CCE16C8B6E
```

## **What's next**

*Learn about additional debugging tools, including:*

- [Logging](#)
- [Monitoring](#)
- [Getting into containers via exec](#)
- [Connecting to containers via proxies](#)
- [Connecting to containers via port forwarding](#)
- [Inspect Kubernetes node with crictl](#)

## **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](#).*

*Last modified May 04, 2020 at 6:56 PM PST: [move setup konnectivity svc \(5fe8c3ca5\)](#)*

[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Using kubectl describe pod to fetch details about pods](#)
- [Example: debugging Pending Pods](#)
- [Example: debugging a down/unreachable node](#)
- [What's next](#)

## **Auditing**

*Kubernetes auditing provides a security-relevant, chronological set of records documenting the sequence of actions in a cluster. The cluster audits the activities generated by users, by applications that use the Kubernetes API, and by the control plane itself.*

*Auditing allows cluster administrators to answer the following questions:*

- *what happened?*
- *when did it happen?*
- *who initiated it?*
- *on what did it happen?*
- *where was it observed?*
- *from where was it initiated?*
- *to where was it going?*

*Audit records begin their lifecycle inside the [kube-apiserver](#) component. Each request on each stage of its execution generates an*

*audit event, which is then pre-processed according to a certain policy and written to a backend. The policy determines what's recorded and the backends persist the records. The current backend implementations include logs files and webhooks.*

*Each request can be recorded with an associated stage. The defined stages are:*

- *RequestReceived - The stage for events generated as soon as the audit handler receives the request, and before it is delegated down the handler chain.*
- *ResponseStarted - Once the response headers are sent, but before the response body is sent. This stage is only generated for long-running requests (e.g. watch).*
- *ResponseComplete - The response body has been completed and no more bytes will be sent.*
- *Panic - Events generated when a panic occurred.*

**Note:** Audit events are different from the [Event API object](#).

*The audit logging feature increases the memory consumption of the API server because some context required for auditing is stored for each request. Memory consumption depends on the audit logging configuration.*

## Audit policy

*Audit policy defines rules about what events should be recorded and what data they should include. The audit policy object structure is defined in the [audit.k8s.io API group](#). When an event is processed, it's compared against the list of rules in order. The first matching rule sets the audit level of the event. The defined audit levels are:*

- *None - don't log events that match this rule.*
- *Metadata - log request metadata (requesting user, timestamp, resource, verb, etc.) but not request or response body.*
- *Request - log event metadata and request body but not response body. This does not apply for non-resource requests.*
- *RequestResponse - log event metadata, request and response bodies. This does not apply for non-resource requests.*

*You can pass a file with the policy to kube-apiserver using the --audit-policy-file flag. If the flag is omitted, no events are logged. Note that the rules field **must** be provided in the audit policy file. A policy with no (0) rules is treated as illegal.*

*Below is an example audit policy file:*

[audit/audit-policy.yaml](#)



```

apiVersion: audit.k8s.io/v1 # This is required.
kind: Policy
# Don't generate audit events for all requests in
RequestReceived stage.
omitStages:
- "RequestReceived"
rules:
# Log pod changes at RequestResponse level
- level: RequestResponse
resources:
- group: ""
  # Resource "pods" doesn't match requests to any
  subresource of pods,
  # which is consistent with the RBAC policy.
  resources: ["pods"]
# Log "pods/log", "pods/status" at Metadata level
- level: Metadata
resources:
- group: ""
  resources: ["pods/log", "pods/status"]

# Don't log requests to a configmap called "controller-leader"
- level: None
resources:
- group: ""
  resources: ["configmaps"]
  resourceNames: ["controller-leader"]

# Don't log watch requests by the "system:kube-proxy" on
endpoints or services
- level: None
users: ["system:kube-proxy"]
verbs: ["watch"]
resources:
- group: "" # core API group
  resources: ["endpoints", "services"]

# Don't log authenticated requests to certain non-resource
URL paths.
- level: None
userGroups: ["system:authenticated"]
nonResourceURLs:
- "/api*" # Wildcard matching.
- "/version"

# Log the request body of configmap changes in kube-system.
- level: Request
resources:
- group: "" # core API group
  resources: ["configmaps"]
# This rule only applies to resources in the "kube-

```

```

system" namespace.
# The empty string "" can be used to select non-
namespaced resources.
namespaces: ["kube-system"]

# Log configmap and secret changes in all other namespaces
at the Metadata level.
- level: Metadata
  resources:
    - group: "" # core API group
      resources: ["secrets", "configmaps"]

# Log all other resources in core and extensions at the
Request level.
- level: Request
  resources:
    - group: "" # core API group
    - group: "extensions" # Version of group should NOT be
included.

# A catch-all rule to log all other requests at the
Metadata level.
- level: Metadata
  # Long-running requests like watches that fall under
  this rule will not
  # generate an audit event in RequestReceived.
  omitStages:
    - "RequestReceived"

```

You can use a minimal audit policy file to log all requests at the `Metadata` level:

```

# Log all requests at the Metadata level.
apiVersion: audit.k8s.io/v1
kind: Policy
rules:
- level: Metadata

```

If you're crafting your own audit profile, you can use the audit profile for Google Container-Optimized OS as a starting point. You can check the [configure-helper.sh](#) script, which generates an audit policy file. You can see most of the audit policy file by looking directly at the script.

## Audit backends

Audit backends persist audit events to an external storage. Out of the box, the kube-apiserver provides two backends:

- Log backend, which writes events into the filesystem
- Webhook backend, which sends events to an external HTTP API

In all cases, audit events follow a structure defined by the Kubernetes API in the `audit.k8s.io` API group. For Kubernetes v1.20.0, that API is at version [v1](#).

**Note:**

In case of patches, request body is a JSON array with patch operations, not a JSON object with an appropriate Kubernetes API object. For example, the following request body is a valid patch request to `/apis/batch/v1/namespaces/some-namespace/jobs/some-job-name`:

```
[  
  {  
    "op": "replace",  
    "path": "/spec/parallelism",  
    "value": 0  
  },  
  {  
    "op": "remove",  
    "path": "/spec/template/spec/containers/0/  
terminationMessagePolicy"  
  }  
]
```

## Log backend

The log backend writes audit events to a file in [JSONlines](#) format. You can configure the log audit backend using the following `kube-apiserver` flags:

- `--audit-log-path` specifies the log file path that log backend uses to write audit events. Not specifying this flag disables log backend. - means standard out
- `--audit-log-maxage` defined the maximum number of days to retain old audit log files
- `--audit-log-maxbackup` defines the maximum number of audit log files to retain
- `--audit-log-maxsize` defines the maximum size in megabytes of the audit log file before it gets rotated

If your cluster's control plane runs the `kube-apiserver` as a Pod, remember to mount the `hostPath` to the location of the policy file and log file, so that audit records are persisted. For example:

```
--audit-policy-file=/etc/kubernetes/audit-policy.yaml \\\n--audit-log-path=/var/log/audit.log
```

then mount the volumes:

```
...
volumeMounts:
- mountPath: /etc/kubernetes/audit-policy.yaml
  name: audit
  readOnly: true
- mountPath: /var/log/audit.log
  name: audit-log
  readOnly: false
```

and finally configure the `hostPath`:

```
...
- name: audit
  hostPath:
    path: /etc/kubernetes/audit-policy.yaml
    type: File

- name: audit-log
  hostPath:
    path: /var/log/audit.log
    type: FileOrCreate
```

## Webhook backend

The webhook audit backend sends audit events to a remote web API, which is assumed to be a form of the Kubernetes API, including means of authentication. You can configure a webhook audit backend using the following `kube-apiserver` flags:

- `--audit-webhook-config-file` specifies the path to a file with a webhook configuration. The webhook configuration is effectively a specialized [kubeconfig](#).
- `--audit-webhook-initial-backoff` specifies the amount of time to wait after the first failed request before retrying. Subsequent requests are retried with exponential backoff.

The webhook config file uses the `kubeconfig` format to specify the remote address of the service and credentials used to connect to it.

## Event batching

Both log and webhook backends support batching. Using webhook as an example, here's the list of available flags. To get the same flag for log backend, replace `webhook` with `log` in the flag name. By default, batching is enabled in webhook and disabled in log. Similarly, by default throttling is enabled in webhook and disabled in log.

- `--audit-webhook-mode` defines the buffering strategy. One of the following:
  - `batch` - buffer events and asynchronously process them in batches. This is the default.

- *blocking* - block API server responses on processing each individual event.
- *blocking-strict* - Same as *blocking*, but when there is a failure during audit logging at the `RequestReceived` stage, the whole request to the `kube-apiserver` fails.

The following flags are used only in the *batch* mode:

- `--audit-webhook-batch-buffer-size` defines the number of events to buffer before batching. If the rate of incoming events overflows the buffer, events are dropped.
- `--audit-webhook-batch-max-size` defines the maximum number of events in one batch.
- `--audit-webhook-batch-max-wait` defines the maximum amount of time to wait before unconditionally batching events in the queue.
- `--audit-webhook-batch-throttle-qps` defines the maximum average number of batches generated per second.
- `--audit-webhook-batch-throttle-burst` defines the maximum number of batches generated at the same moment if the allowed QPS was underutilized previously.

## **Parameter tuning**

Parameters should be set to accommodate the load on the API server.

For example, if `kube-apiserver` receives 100 requests each second, and each request is audited only on `ResponseStarted` and `ResponseComplete` stages, you should account for ~200 audit events being generated each second. Assuming that there are up to 100 events in a batch, you should set throttling level at least 2 queries per second. Assuming that the backend can take up to 5 seconds to write events, you should set the buffer size to hold up to 5 seconds of events; that is: 10 batches, or 1000 events.

In most cases however, the default parameters should be sufficient and you don't have to worry about setting them manually. You can look at the following Prometheus metrics exposed by `kube-apiserver` and in the logs to monitor the state of the auditing subsystem.

- `apiserver_audit_event_total` metric contains the total number of audit events exported.
- `apiserver_audit_error_total` metric contains the total number of events dropped due to an error during exporting.

## **Log entry truncation**

Both log and webhook backends support limiting the size of events that are logged. As an example, the following is the list of flags available for the log backend:

- `audit-log-truncate-enabled` whether event and batch truncating is enabled.
- `audit-log-truncate-max-batch-size` maximum size in bytes of the batch sent to the underlying backend.
- `audit-log-truncate-max-event-size` maximum size in bytes of the audit event sent to the underlying backend.

By default truncate is disabled in both webhook and log, a cluster administrator should set `audit-log-truncate-enabled` or `audit-webhook-truncate-enabled` to enable the feature.

## **What's next**

- Learn about [Mutating webhook auditing annotations](#).

## **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](#).

Last modified August 09, 2020 at 12:52 PM PST: [Update cluster auditing task page \(304661b15\)](#)

[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Audit policy](#)
- [Audit backends](#)
  - [Log backend](#)
  - [Webhook backend](#)
- [Event batching](#)
- [Parameter tuning](#)
  - [Log entry truncation](#)
- [What's next](#)

## **Debug a StatefulSet**

This task shows you how to debug a StatefulSet.

## **Before you begin**

- You need to have a Kubernetes cluster, and the `kubectl` command-line tool must be configured to communicate with your cluster.
- You should have a `StatefulSet` running that you want to investigate.

## **Debugging a StatefulSet**

In order to list all the pods which belong to a `StatefulSet`, which have a label `app=myapp` set on them, you can use the following:

```
kubectl get pods -l app=myapp
```

If you find that any Pods listed are in `Unknown` or `Terminating` state for an extended period of time, refer to the [Deleting StatefulSet Pods](#) task for instructions on how to deal with them. You can debug individual Pods in a `StatefulSet` using the [Debugging Pods](#) guide.

## **What's next**

Learn more about [debugging an init-container](#).

## **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](#).

Last modified August 07, 2020 at 4:46 PM PST: [Tune links in tasks section \(1/2\) \(b8541d212\)](#)

[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Debugging a StatefulSet](#)
- [What's next](#)

## **Debug Init Containers**

This page shows how to investigate problems related to the execution of Init Containers. The example command lines below refer to the Pod as `<pod-name>` and the Init Containers as `<init-container-1>` and `<init-container-2>`.

## **Before you begin**

You need to have a Kubernetes cluster, and the `kubectl` command-line tool must be configured to communicate with your cluster. If you do not already have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:

- [Katacoda](#)
- [Play with Kubernetes](#)

To check the version, enter `kubectl version`.

- You should be familiar with the basics of [Init Containers](#).
- You should have [Configured an Init Container](#).

## **Checking the status of Init Containers**

Display the status of your pod:

```
kubectl get pod <pod-name>
```

For example, a status of `Init:1/2` indicates that one of two Init Containers has completed successfully:

NAME	READY	STATUS	RESTARTS	AGE
<pod-name>	0/1	Init:1/2	0	7s

See [Understanding Pod status](#) for more examples of status values and their meanings.

## **Getting details about Init Containers**

View more detailed information about Init Container execution:

```
kubectl describe pod <pod-name>
```

For example, a Pod with two Init Containers might show the following:

```
Init Containers:  
<init-container-1>:  
  Container ID: ...  
  ...  
  State:          Terminated  
  Reason:        Completed
```

```
Exit Code:      0
Started:       ...
Finished:      ...
Ready:         True
Restart Count: 0
...
<init-container-2>:
Container ID: ...
...
State:          Waiting
Reason:         CrashLoopBackOff
Last State:    Terminated
Reason:         Error
Exit Code:     1
Started:       ...
Finished:      ...
Ready:         False
Restart Count: 3
...
```

You can also access the Init Container statuses programmatically by reading the `status.initContainerStatuses` field on the Pod Spec:

```
kubectl get pod nginx --template '{{.status.initContainerStatuses}}'
```

This command will return the same information as above in raw JSON.

## Accessing logs from Init Containers

Pass the Init Container name along with the Pod name to access its logs.

```
kubectl logs <pod-name> -c <init-container-2>
```

Init Containers that run a shell script print commands as they're executed. For example, you can do this in Bash by running `set -x` at the beginning of the script.

## Understanding Pod status

A Pod status beginning with `Init:` summarizes the status of Init Container execution. The table below describes some example status values that you might see while debugging Init Containers.

Status	Meaning
<code>Init:N/M</code>	The Pod has $M$ Init Containers, and $N$ have completed so far.
<code>Init:Error</code>	An Init Container has failed to execute.
<code>Init:CrashLoopBackOff</code>	An Init Container has failed repeatedly.
<code>Pending</code>	The Pod has not yet begun executing Init Containers.
<code>PodInitializing or Running</code>	The Pod has already finished executing Init Containers.

## 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](#).

Last modified August 07, 2020 at 4:46 PM PST: [Tune links in tasks section \(1/2\) \(b8541d212\)](#)

[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Checking the status of Init Containers](#)
- [Getting details about Init Containers](#)
- [Accessing logs from Init Containers](#)
- [Understanding Pod status](#)

# Debug Pods and ReplicationControllers

This page shows how to debug Pods and ReplicationControllers.

## Before you begin

You need to have a Kubernetes cluster, and the `kubectl` command-line tool must be configured to communicate with your cluster. If you do not already have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:

- [Katacoda](#)
- [Play with Kubernetes](#)

To check the version, enter `kubectl version`.

- You should be familiar with the basics of [Pods](#) and with Pods' [lifecycles](#).

## Debugging Pods

The first step in debugging a pod is taking a look at it. Check the current state of the pod and recent events with the following command:

```
kubectl describe pods ${POD_NAME}
```

Look at the state of the containers in the pod. Are they all *Running*? Have there been recent restarts?

Continue debugging depending on the state of the pods.

### My pod stays pending

If a pod is stuck in *Pending* it means that it can not be scheduled onto a node. Generally this is because there are insufficient resources of one type or another that prevent scheduling. Look at the output of the `kubectl describe ...` command above. There should be messages from the scheduler about why it can not schedule your pod. Reasons include:

#### Insufficient resources

You may have exhausted the supply of CPU or Memory in your cluster. In this case you can try several things:

- Add more nodes to the cluster.
- [Terminate unneeded pods](#) to make room for pending pods.
- Check that the pod is not larger than your nodes. For example, if all nodes have a capacity of `cpu: 1`, then a pod with a request of `cpu: 1.1` will never be scheduled.

You can check node capacities with the `kubectl get nodes -o <format>` command. Here are some example command lines that extract just the necessary information:

```
kubectl get nodes -o yaml | egrep '\sname:/cpu:/memory:'  
kubectl get nodes -o json | jq '.items[] |  
{name: .metadata.name, cap: .status.capacity}'
```

The [resource quota](#) feature can be configured to limit the total amount of resources that can be consumed. If used in conjunction with namespaces, it can prevent one team from hogging all the resources.

## **Using hostPort**

When you bind a pod to a `hostPort` there are a limited number of places that the pod can be scheduled. In most cases, `hostPort` is unnecessary; try using a service object to expose your pod. If you do require `hostPort` then you can only schedule as many pods as there are nodes in your container cluster.

## **My pod stays waiting**

If a pod is stuck in the `Waiting` state, then it has been scheduled to a worker node, but it can't run on that machine. Again, the information from `kubectl describe ...` should be informative. The most common cause of `Waiting` pods is a failure to pull the image. There are three things to check:

- Make sure that you have the name of the image correct.
- Have you pushed the image to the repository?
- Run a manual `docker pull <image>` on your machine to see if the image can be pulled.

## **My pod is crashing or otherwise unhealthy**

Once your pod has been scheduled, the methods described in [Debug Running Pods](#) are available for debugging.

## **Debugging ReplicationControllers**

ReplicationControllers are fairly straightforward. They can either create pods or they can't. If they can't create pods, then please refer to the [instructions above](#) to debug your pods.

You can also use `kubectl describe rc ${CONTROLLER_NAME}` to inspect events related to the replication controller.

# 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](#).

Last modified December 15, 2020 at 9:46 PM PST: [Update debug-pod-replication-controller.md \(8ca4d7d74\)](#)

[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Debugging Pods](#)
  - [My pod stays pending](#)
  - [My pod stays waiting](#)
  - [My pod is crashing or otherwise unhealthy](#)
- [Debugging ReplicationControllers](#)

# Debug Running Pods

This page explains how to debug Pods running (or crashing) on a Node.

## Before you begin

- Your [Pod](#) should already be scheduled and running. If your Pod is not yet running, start with [Troubleshoot Applications](#).
- For some of the advanced debugging steps you need to know on which Node the Pod is running and have shell access to run commands on that Node. You don't need that access to run the standard debug steps that use `kubectl`.

## Examining pod logs

First, look at the logs of the affected container:

```
kubectl logs ${POD_NAME} ${CONTAINER_NAME}
```

If your container has previously crashed, you can access the previous container's crash log with:

```
kubectl logs --previous ${POD_NAME} ${CONTAINER_NAME}
```

## **Debugging with container exec**

If the [container image](#) includes debugging utilities, as is the case with images built from Linux and Windows OS base images, you can run commands inside a specific container with `kubectl exec`:

```
kubectl exec ${POD_NAME} -c ${CONTAINER_NAME} -- ${CMD} ${ARG1} ${ARG2} ... ${ARGN}
```

**Note:** `-c ${CONTAINER_NAME}` is optional. You can omit it for Pods that only contain a single container.

As an example, to look at the logs from a running Cassandra pod, you might run

```
kubectl exec cassandra -- cat /var/log/cassandra/system.log
```

You can run a shell that's connected to your terminal using the `-i` and `-t` arguments to `kubectl exec`, for example:

```
kubectl exec -it cassandra -- sh
```

For more details, see [Get a Shell to a Running Container](#).

## **Debugging with an ephemeral debug container**

**FEATURE STATE:** Kubernetes v1.18 [alpha]

[Ephemeral containers](#) are useful for interactive troubleshooting when `kubectl exec` is insufficient because a container has crashed or a container image doesn't include debugging utilities, such as with [distroless images](#). `kubectl` has an alpha command that can create ephemeral containers for debugging beginning with version v1.18.

### **Example debugging using ephemeral containers**

**Note:** The examples in this section require the `EphemeralContainers` [feature gate](#) enabled in your cluster and `kubectl` version v1.18 or later.

You can use the `kubectl debug` command to add ephemeral containers to a running Pod. First, create a pod for the example:

```
kubectl run ephemeral-demo --image=k8s.gcr.io/pause:3.1 --restart=Never
```

The examples in this section use the `pause` container image because it does not contain userland debugging utilities, but this method works with all container images.

If you attempt to use `kubectl exec` to create a shell you will see an error because there is no shell in this container image.

```
kubectl exec -it ephemeral-demo -- sh
```

```
OCI runtime exec failed: exec failed: container_linux.go:346: starting container process caused "exec: \"sh\": executable file not found in $PATH": unknown
```

You can instead add a debugging container using `kubectl debug`. If you specify the `-i/- --interactive` argument, `kubectl` will automatically attach to the console of the Ephemeral Container.

```
kubectl debug -it ephemeral-demo --image=busybox --target=ephemeral-demo
```

```
Defaulting debug container name to debugger-8xzrl.  
If you don't see a command prompt, try pressing enter.  
/ #
```

This command adds a new busybox container and attaches to it. The `--target` parameter targets the process namespace of another container. It's necessary here because `kubectl run` does not enable [process namespace sharing](#) in the pod it creates.

**Note:** The `--target` parameter must be supported by the [Container Runtime](#). When not supported, the Ephemeral Container may not be started, or it may be started with an isolated process namespace.

You can view the state of the newly created ephemeral container using `kubectl describe`:

```
kubectl describe pod ephemeral-demo
```

```
...  
Ephemeral Containers:  
  debugger-8xzrl:  
    Container ID: docker://  
    b888f9adfd15bd5739fefaa39e1df4dd3c617b9902082b1cfdc29c4028ffb  
    2eb  
      Image:          busybox  
      Image ID:       docker-pullable://  
      busybox@sha256:1828edd60c5efd34b2bf5dd3282ec0cc04d47b2ff9caa0  
      b6d4f07a21d1c08084  
      Port:           <none>  
      Host Port:     <none>  
      State:          Running  
      Started:        Wed, 12 Feb 2020 14:25:42 +0100  
      Ready:           False  
      Restart Count:  0  
      Environment:    <none>
```

```
Mounts:           <none>
```

```
...
```

Use `kubectl delete` to remove the Pod when you're finished:

```
kubectl delete pod ephemeral-demo
```

## **Debugging using a copy of the Pod**

Sometimes Pod configuration options make it difficult to troubleshoot in certain situations. For example, you can't run `kubectl exec` to troubleshoot your container if your container image does not include a shell or if your application crashes on startup. In these situations you can use `kubectl debug` to create a copy of the Pod with configuration values changed to aid debugging.

### **Copying a Pod while adding a new container**

Adding a new container can be useful when your application is running but not behaving as you expect and you'd like to add additional troubleshooting utilities to the Pod.

For example, maybe your application's container images are built on `busybox` but you need debugging utilities not included in `busybox`. You can simulate this scenario using `kubectl run`:

```
kubectl run myapp --image=busybox --restart=Never -- sleep 1d
```

Run this command to create a copy of `myapp` named `myapp-copy` that adds a new Ubuntu container for debugging:

```
kubectl debug myapp -it --image=ubuntu --share-processes --copy-to=myapp-debug
```

Defaulting debug container name to `debugger-w7xmf`.  
If you don't see a command prompt, try pressing enter.  
`root@myapp-debug:/#`

#### **Note:**

- `kubectl debug` automatically generates a container name if you don't choose one using the `--container` flag.
- The `-i` flag causes `kubectl debug` to attach to the new container by default. You can prevent this by specifying `--attach=false`. If your session becomes disconnected you can reattach using `kubectl attach`.
- The `--share-processes` allows the containers in this Pod to see processes from the other containers in the Pod. For more information about how this works, see [Share Process Namespace between Containers in a Pod](#).

Don't forget to clean up the debugging Pod when you're finished with it:

```
kubectl delete pod myapp myapp-debug
```

## **Copying a Pod while changing its command**

*Sometimes it's useful to change the command for a container, for example to add a debugging flag or because the application is crashing.*

*To simulate a crashing application, use `kubectl run` to create a container that immediately exits:*

```
kubectl run --image=busybox myapp -- false
```

*You can see using `kubectl describe pod myapp` that this container is crashing:*

*Containers:*

<i>myapp:</i>	
<i>  Image:</i>	<i>busybox</i>
<i>  ...</i>	
<i>  Args:</i>	
<i>    false</i>	
<i>  State:</i>	<i>Waiting</i>
<i>    Reason:</i>	<i>CrashLoopBackOff</i>
<i>  Last State:</i>	<i>Terminated</i>
<i>    Reason:</i>	<i>Error</i>
<i>  Exit Code:</i>	<i>1</i>

*You can use `kubectl debug` to create a copy of this Pod with the command changed to an interactive shell:*

```
kubectl debug myapp -it --copy-to=myapp-debug --  
container=myapp -- sh
```

*If you don't see a command prompt, try pressing enter.  
/ #*

*Now you have an interactive shell that you can use to perform tasks like checking filesystem paths or running the container command manually.*

### **Note:**

- To change the command of a specific container you must specify its name using `--container` or `kubectl debug` will instead create a new container to run the command you specified.
- The `-i` flag causes `kubectl debug` to attach to the container by default. You can prevent this by specifying `--attach=false`. If your session becomes disconnected you can reattach using `kubectl attach`.

*Don't forget to clean up the debugging Pod when you're finished with it:*

```
kubectl delete pod myapp myapp-debug
```

## **Copying a Pod while changing container images**

*In some situations you may want to change a misbehaving Pod from its normal production container images to an image containing a debugging build or additional utilities.*

*As an example, create a Pod using kubectl run:*

```
kubectl run myapp --image=busybox --restart=Never -- sleep 1d
```

*Now use kubectl debug to make a copy and change its container image to ubuntu:*

```
kubectl debug myapp --copy-to=myapp-debug --set-image=*=ubuntu
```

*The syntax of --set-image uses the same container\_name=image syntax as kubectl set image. \*=ubuntu means change the image of all containers to ubuntu.*

*Don't forget to clean up the debugging Pod when you're finished with it:*

```
kubectl delete pod myapp myapp-debug
```

## **Debugging via a shell on the node**

*If none of these approaches work, you can find the Node on which the Pod is running and create a privileged Pod running in the host namespaces. To create an interactive shell on a node using kubectl debug, run:*

```
kubectl debug node/mynode -it --image=ubuntu
```

*Creating debugging pod node-debugger-mynode-pdx84 with container debugger on node mynode.*

*If you don't see a command prompt, try pressing enter.  
root@ek8s:/#*

*When creating a debugging session on a node, keep in mind that:*

- *kubectl debug automatically generates the name of the new Pod based on the name of the Node.*
- *The container runs in the host IPC, Network, and PID namespaces.*
- *The root filesystem of the Node will be mounted at /host.*

*Don't forget to clean up the debugging Pod when you're finished with it:*

```
kubectl delete pod node-debugger-mynode-pdx84
```

## **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](#).

Last modified December 07, 2020 at 2:02 PM PST: [Fix typo in Debug Running Pods task \(09665e40e\)](#)

[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Examining pod logs](#)
- [Debugging with container exec](#)
- [Debugging with an ephemeral debug container](#)
  - [Example debugging using ephemeral containers](#)
- [Debugging using a copy of the Pod](#)
  - [Copying a Pod while adding a new container](#)
  - [Copying a Pod while changing its command](#)
  - [Copying a Pod while changing container images](#)
- [Debugging via a shell on the node](#)

# Debug Services

An issue that comes up rather frequently for new installations of Kubernetes is that a Service is not working properly. You've run your Pods through a Deployment (or other workload controller) and created a Service, but you get no response when you try to access it. This document will hopefully help you to figure out what's going wrong.

## Running commands in a Pod

For many steps here you will want to see what a Pod running in the cluster sees. The simplest way to do this is to run an interactive alpine Pod:

```
kubectl run -it --rm --restart=Never alpine --image=alpine sh
```

**Note:** If you don't see a command prompt, try pressing enter.

If you already have a running Pod that you prefer to use, you can run a command in it using:

```
kubectl exec <POD-NAME> -c <CONTAINER-NAME> -- <COMMAND>
```

## Setup

For the purposes of this walk-through, let's run some Pods. Since you're probably debugging your own Service you can substitute your own details, or you can follow along and get a second data point.

```
kubectl create deployment hostnames --image=k8s.gcr.io/serve_hostname
```

```
deployment.apps/hostnames created
```

*kubectl commands will print the type and name of the resource created or mutated, which can then be used in subsequent commands.*

*Let's scale the deployment to 3 replicas.*

```
kubectl scale deployment hostnames --replicas=3
```

```
deployment.apps/hostnames scaled
```

*Note that this is the same as if you had started the Deployment with the following YAML:*

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: hostnames
  name: hostnames
spec:
  selector:
    matchLabels:
      app: hostnames
  replicas: 3
  template:
    metadata:
      labels:
        app: hostnames
    spec:
      containers:
        - name: hostnames
          image: k8s.gcr.io/serve_hostname
```

*The label "app" is automatically set by `kubectl create deployment` to the name of the Deployment.*

*You can confirm your Pods are running:*

```
kubectl get pods -l app=hostnames
```

NAME	READY	STATUS	RESTARTS	
AGE				
hostnames-632524106-bbpiw	1/1	Running	0	2m
hostnames-632524106-ly40y	1/1	Running	0	2m
hostnames-632524106-tlaok	1/1	Running	0	2m

*You can also confirm that your Pods are serving. You can get the list of Pod IP addresses and test them directly.*

```
kubectl get pods -l app=hostnames \
-o go-template='{{range .items}}{{.status.podIP}}{{"\n"}}{{end}}'
```

```
10.244.0.5
10.244.0.6
10.244.0.7
```

The example container used for this walk-through simply serves its own hostname via HTTP on port 9376, but if you are debugging your own app, you'll want to use whatever port number your Pods are listening on.

From within a pod:

```
for ep in 10.244.0.5:9376 10.244.0.6:9376 10.244.0.7:9376; do
    wget -qO- $ep
done
```

This should produce something like:

```
hostnames-632524106-bbpiw
hostnames-632524106-ly40y
hostnames-632524106-tlaok
```

If you are not getting the responses you expect at this point, your Pods might not be healthy or might not be listening on the port you think they are. You might find `kubectl logs` to be useful for seeing what is happening, or perhaps you need to `kubectl exec` directly into your Pods and debug from there.

Assuming everything has gone to plan so far, you can start to investigate why your Service doesn't work.

## **Does the Service exist?**

The astute reader will have noticed that you did not actually create a Service yet - that is intentional. This is a step that sometimes gets forgotten, and is the first thing to check.

What would happen if you tried to access a non-existent Service? If you have another Pod that consumes this Service by name you would get something like:

```
wget -O- hostnames
```

```
Resolving hostnames (hostnames)... failed: Name or service not known.
```

```
wget: unable to resolve host address 'hostnames'
```

The first thing to check is whether that Service actually exists:

```
kubectl get svc hostnames
```

```
No resources found.
```

```
Error from server (NotFound): services "hostnames" not found
```

*Let's create the Service. As before, this is for the walk-through - you can use your own Service's details here.*

```
kubectl expose deployment hostnames --port=80 --target-port=9376
```

```
service/hostnames exposed
```

*And read it back, just to be sure:*

```
kubectl get svc hostnames
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)
hostnames	ClusterIP	10.0.1.175	<none>	80/TCP

5s

*Now you know that the Service exists.*

*As before, this is the same as if you had started the Service with YAML:*

```
apiVersion: v1
kind: Service
metadata:
  labels:
    app: hostnames
    name: hostnames
spec:
  selector:
    app: hostnames
  ports:
  - name: default
    protocol: TCP
    port: 80
    targetPort: 9376
```

*In order to highlight the full range of configuration, the Service you created here uses a different port number than the Pods. For many real-world Services, these values might be the same.*

## **Does the Service work by DNS name?**

*One of the most common ways that clients consume a Service is through a DNS name.*

*From a Pod in the same Namespace:*

```
nslookup hostnames
```

```
Address 1: 10.0.0.10 kube-dns.kube-system.svc.cluster.local
```

```
Name: hostnames
```

```
Address 1: 10.0.1.175 hostnames.default.svc.cluster.local
```

If this fails, perhaps your Pod and Service are in different Namespaces, try a namespace-qualified name (again, from within a Pod):

```
nslookup hostnames.default
```

```
Address 1: 10.0.0.10 kube-dns.kube-system.svc.cluster.local
```

```
Name: hostnames.default
```

```
Address 1: 10.0.1.175 hostnames.default.svc.cluster.local
```

If this works, you'll need to adjust your app to use a cross-namespace name, or run your app and Service in the same Namespace. If this still fails, try a fully-qualified name:

```
nslookup hostnames.default.svc.cluster.local
```

```
Address 1: 10.0.0.10 kube-dns.kube-system.svc.cluster.local
```

```
Name: hostnames.default.svc.cluster.local
```

```
Address 1: 10.0.1.175 hostnames.default.svc.cluster.local
```

Note the suffix here: "default.svc.cluster.local". The "default" is the Namespace you're operating in. The "svc" denotes that this is a Service. The "cluster.local" is your cluster domain, which COULD be different in your own cluster.

You can also try this from a Node in the cluster:

**Note:** 10.0.0.10 is the cluster's DNS Service IP, yours might be different.

```
nslookup hostnames.default.svc.cluster.local 10.0.0.10
```

```
Server: 10.0.0.10
```

```
Address: 10.0.0.10#53
```

```
Name: hostnames.default.svc.cluster.local
```

```
Address: 10.0.1.175
```

If you are able to do a fully-qualified name lookup but not a relative one, you need to check that your /etc/resolv.conf file in your Pod is correct. From within a Pod:

```
cat /etc/resolv.conf
```

You should see something like:

```
nameserver 10.0.0.10
```

```
search default.svc.cluster.local svc.cluster.local
```

```
cluster.local example.com
options ndots:5
```

The `nameserver` line must indicate your cluster's DNS Service. This is passed into `kubelet` with the `--cluster-dns` flag.

The `search` line must include an appropriate suffix for you to find the Service name. In this case it is looking for Services in the local Namespace ("`default.svc.cluster.local`"), Services in all Namespaces ("`svc.cluster.local`"), and lastly for names in the cluster ("`cluster.local`"). Depending on your own install you might have additional records after that (up to 6 total). The cluster suffix is passed into `kubelet` with the `--cluster-domain` flag. Throughout this document, the cluster suffix is assumed to be "`cluster.local`". Your own clusters might be configured differently, in which case you should change that in all of the previous commands.

The `options` line must set `ndots` high enough that your DNS client library considers search paths at all. Kubernetes sets this to 5 by default, which is high enough to cover all of the DNS names it generates.

## Does any Service work by DNS name?

If the above still fails, DNS lookups are not working for your Service. You can take a step back and see what else is not working. The Kubernetes master Service should always work. From within a Pod:

```
nslookup kubernetes.default
```

```
Server: 10.0.0.10
Address 1: 10.0.0.10 kube-dns.kube-system.svc.cluster.local

Name: kubernetes.default
Address 1: 10.0.0.1 kubernetes.default.svc.cluster.local
```

If this fails, please see the [kube-proxy](#) section of this document, or even go back to the top of this document and start over, but instead of debugging your own Service, debug the DNS Service.

## Does the Service work by IP?

Assuming you have confirmed that DNS works, the next thing to test is whether your Service works by its IP address. From a Pod in your cluster, access the Service's IP (from `kubectl get` above).

```
for i in $(seq 1 3); do
    wget -qO- 10.0.1.175:80
done
```

This should produce something like:

```
hostnames-632524106-bbpiw
hostnames-632524106-ly40y
hostnames-632524106-tlaok
```

If your Service is working, you should get correct responses. If not, there are a number of things that could be going wrong. Read on.

## Is the Service defined correctly?

It might sound silly, but you should really double and triple check that your Service is correct and matches your Pod's port. Read back your Service and verify it:

```
kubectl get service hostnames -o json
```

```
{
  "kind": "Service",
  "apiVersion": "v1",
  "metadata": {
    "name": "hostnames",
    "namespace": "default",
    "uid": "428c8b6c-24bc-11e5-936d-42010af0a9bc",
    "resourceVersion": "347189",
    "creationTimestamp": "2015-07-07T15:24:29Z",
    "labels": {
      "app": "hostnames"
    }
  },
  "spec": {
    "ports": [
      {
        "name": "default",
        "protocol": "TCP",
        "port": 80,
        "targetPort": 9376,
        "nodePort": 0
      }
    ],
    "selector": {
      "app": "hostnames"
    },
    "clusterIP": "10.0.1.175",
    "type": "ClusterIP",
    "sessionAffinity": "None"
  },
  "status": {
    "loadBalancer": {}
  }
}
```

- Is the Service port you are trying to access listed in `spec.ports[]`?

- Is the `targetPort` correct for your Pods (some Pods use a different port than the Service)?
- If you meant to use a numeric port, is it a number (9376) or a string "9376"?
- If you meant to use a named port, do your Pods expose a port with the same name?
- Is the port's protocol correct for your Pods?

## **Does the Service have any Endpoints?**

If you got this far, you have confirmed that your Service is correctly defined and is resolved by DNS. Now let's check that the Pods you ran are actually being selected by the Service.

Earlier you saw that the Pods were running. You can re-check that:

```
kubectl get pods -l app=hostnames
```

NAME	READY	STATUS	RESTARTS	AGE
hostnames-632524106-bbpiw	1/1	Running	0	1h
hostnames-632524106-ly40y	1/1	Running	0	1h
hostnames-632524106-tlaok	1/1	Running	0	1h

The `-l app=hostnames` argument is a label selector - just like our Service has.

The "AGE" column says that these Pods are about an hour old, which implies that they are running fine and not crashing.

The "RESTARTS" column says that these pods are not crashing frequently or being restarted. Frequent restarts could lead to intermittent connectivity issues. If the restart count is high, read more about how to [debug pods](#).

Inside the Kubernetes system is a control loop which evaluates the selector of every Service and saves the results into a corresponding Endpoints object.

```
kubectl get endpoints hostnames
```

NAME	ENDPOINTS
hostnames	10.244.0.5:9376,10.244.0.6:9376,10.244.0.7:9376

This confirms that the endpoints controller has found the correct Pods for your Service. If the `ENDPOINTS` column is `<none>`, you should check that the `spec.selector` field of your Service actually selects for `metaData.labels` values on your Pods. A common mistake is to have a typo or other error, such as the Service selecting for `app=hostnames`, but the Deployment specifying `run=hostnames`, as in versions previous to 1.18, where the `kubectl run` command could have been also used to create a Deployment.

## **Are the Pods working?**

*At this point, you know that your Service exists and has selected your Pods. At the beginning of this walk-through, you verified the Pods themselves. Let's check again that the Pods are actually working - you can bypass the Service mechanism and go straight to the Pods, as listed by the Endpoints above.*

**Note:** These commands use the Pod port (9376), rather than the Service port (80).

*From within a Pod:*

```
for ep in 10.244.0.5:9376 10.244.0.6:9376 10.244.0.7:9376; do
    wget -qO- $ep
done
```

*This should produce something like:*

```
hostnames-632524106-bbpiw
hostnames-632524106-ly40y
hostnames-632524106-tlaok
```

*You expect each Pod in the Endpoints list to return its own hostname. If this is not what happens (or whatever the correct behavior is for your own Pods), you should investigate what's happening there.*

## **Is the kube-proxy working?**

*If you get here, your Service is running, has Endpoints, and your Pods are actually serving. At this point, the whole Service proxy mechanism is suspect. Let's confirm it, piece by piece.*

*The default implementation of Services, and the one used on most clusters, is kube-proxy. This is a program that runs on every node and configures one of a small set of mechanisms for providing the Service abstraction. If your cluster does not use kube-proxy, the following sections will not apply, and you will have to investigate whatever implementation of Services you are using.*

### **Is kube-proxy running?**

*Confirm that kube-proxy is running on your Nodes. Running directly on a Node, you should get something like the below:*

```
ps auxw | grep kube-proxy
```

```
root 4194 0.4 0.1 101864 17696 ? Sl Jul04 25:43 /usr/local/bin/kube-proxy --master=https://kubernetes-master --kubeconfig=/var/lib/kube-proxy/kubeconfig --v=2
```

Next, confirm that it is not failing something obvious, like contacting the master. To do this, you'll have to look at the logs. Accessing the logs depends on your Node OS. On some OSes it is a file, such as `/var/log/kube-proxy.log`, while other OSes use `journalctl` to access logs. You should see something like:

```
I1027 22:14:53.995134    5063 server.go:200] Running in  
resource-only container "/kube-proxy"  
I1027 22:14:53.998163    5063 server.go:247] Using iptables  
Proxier.  
I1027 22:14:53.999055    5063 server.go:255] Tearing down  
userspace rules. Errors here are acceptable.  
I1027 22:14:54.038140    5063 proxier.go:352] Setting  
endpoints for "kube-system/kube-dns:dns-tcp" to  
[10.244.1.3:53]  
I1027 22:14:54.038164    5063 proxier.go:352] Setting  
endpoints for "kube-system/kube-dns:dns" to [10.244.1.3:53]  
I1027 22:14:54.038209    5063 proxier.go:352] Setting  
endpoints for "default/kubernetes:https" to [10.240.0.2:443]  
I1027 22:14:54.038238    5063 proxier.go:429] Not syncing  
iptables until Services and Endpoints have been received  
from master  
I1027 22:14:54.040048    5063 proxier.go:294] Adding new  
service "default/kubernetes:https" at 10.0.0.1:443/TCP  
I1027 22:14:54.040154    5063 proxier.go:294] Adding new  
service "kube-system/kube-dns:dns" at 10.0.0.10:53/UDP  
I1027 22:14:54.040223    5063 proxier.go:294] Adding new  
service "kube-system/kube-dns:dns-tcp" at 10.0.0.10:53/TCP
```

If you see error messages about not being able to contact the master, you should double-check your Node configuration and installation steps.

One of the possible reasons that `kube-proxy` cannot run correctly is that the required `conntrack` binary cannot be found. This may happen on some Linux systems, depending on how you are installing the cluster, for example, you are installing Kubernetes from scratch. If this is the case, you need to manually install the `conntrack` package (e.g. `sudo apt install conntrack` on Ubuntu) and then retry.

Kube-proxy can run in one of a few modes. In the log listed above, the line `Using iptables Proxier` indicates that `kube-proxy` is running in "iptables" mode. The most common other mode is "ipvs". The older "userspace" mode has largely been replaced by these.

## Iptables mode

In "iptables" mode, you should see something like the following on a Node:

```
iptables-save | grep hostnames
```

```
-A KUBE-SEP-57KPRZ3JQVENLNBR -s 10.244.3.6/32 -m comment --comment "default/hostnames:" -j MARK --set-xmark 0x00004000/0x00004000
-A KUBE-SEP-57KPRZ3JQVENLNBR -p tcp -m comment --comment "default/hostnames:" -m tcp -j DNAT --to-destination 10.244.3.6:9376
-A KUBE-SEP-WNBA2IHDGP2B0BGZ -s 10.244.1.7/32 -m comment --comment "default/hostnames:" -j MARK --set-xmark 0x00004000/0x00004000
-A KUBE-SEP-WNBA2IHDGP2B0BGZ -p tcp -m comment --comment "default/hostnames:" -m tcp -j DNAT --to-destination 10.244.1.7:9376
-A KUBE-SEP-X3P2623AGDH6CDF3 -s 10.244.2.3/32 -m comment --comment "default/hostnames:" -j MARK --set-xmark 0x00004000/0x00004000
-A KUBE-SEP-X3P2623AGDH6CDF3 -p tcp -m comment --comment "default/hostnames:" -m tcp -j DNAT --to-destination 10.244.2.3:9376
-A KUBE-SERVICES -d 10.0.1.175/32 -p tcp -m comment --comment "default/hostnames: cluster IP" -m tcp --dport 80 -j KUBE-SVC-NWV5X2332I40T4T3
-A KUBE-SVC-NWV5X2332I40T4T3 -m comment --comment "default/hostnames:" -m statistic --mode random --probability 0.33332999982 -j KUBE-SEP-WNBA2IHDGP2B0BGZ
-A KUBE-SVC-NWV5X2332I40T4T3 -m comment --comment "default/hostnames:" -m statistic --mode random --probability 0.50000000000 -j KUBE-SEP-X3P2623AGDH6CDF3
-A KUBE-SVC-NWV5X2332I40T4T3 -m comment --comment "default/hostnames:" -j KUBE-SEP-57KPRZ3JQVENLNBR
```

*For each port of each Service, there should be 1 rule in KUBE-SERVICES and one KUBE-SVC-<hash> chain. For each Pod endpoint, there should be a small number of rules in that KUBE-SVC-<hash> and one KUBE-SEP-<hash> chain with a small number of rules in it. The exact rules will vary based on your exact config (including node-ports and load-balancers).*

## *IPVS mode*

In "ipvs" mode, you should see something like the following on a Node:

*ipvsadm -ln*

```

Prot LocalAddress:Port Scheduler Flags
    -> RemoteAddress:Port            Forward Weight ActiveConn
InActConn
...
TCP  10.0.1.175:80 rr
    -> 10.244.0.5:9376              Masq      1      0
0
    -> 10.244.0.6:9376              Masq      1      0
0

```

-> 10.244.0.7:9376	Masq	1	0
0			
...			

For each port of each Service, plus any NodePorts, external IPs, and load-balancer IPs, kube-proxy will create a virtual server. For each Pod endpoint, it will create corresponding real servers. In this example, service hostnames(10.0.1.175:80) has 3 endpoints(10.244.0.5:9376, 10.244.0.6:9376, 10.244.0.7:9376).

## **Userspace mode**

In rare cases, you may be using "userspace" mode. From your Node:

```
iptables-save | grep hostnames
```

```
-A KUBE-PORTALS-CONTAINER -d 10.0.1.175/32 -p tcp -m comment --comment "default/hostnames:default" -m tcp --dport 80 -j REDIRECT --to-ports 48577
-A KUBE-PORTALS-HOST -d 10.0.1.175/32 -p tcp -m comment --comment "default/hostnames:default" -m tcp --dport 80 -j DNAT --to-destination 10.240.115.247:48577
```

There should be 2 rules for each port of your Service (just one in this example) - a "KUBE-PORTALS-CONTAINER" and a "KUBE-PORTALS-HOST".

Almost nobody should be using the "userspace" mode any more, so you won't spend more time on it here.

## **Is kube-proxy proxying?**

Assuming you do see one the above cases, try again to access your Service by IP from one of your Nodes:

```
curl 10.0.1.175:80
```

```
hostnames-632524106-bbpiw
```

If this fails and you are using the userspace proxy, you can try accessing the proxy directly. If you are using the iptables proxy, skip this section.

Look back at the `iptables-save` output above, and extract the port number that kube-proxy is using for your Service. In the above examples it is "48577". Now connect to that:

```
curl localhost:48577
```

```
hostnames-632524106-tlaok
```

If this still fails, look at the `kube-proxy` logs for specific lines like:

```
Setting endpoints for default/hostnames:default to  
[10.244.0.5:9376 10.244.0.6:9376 10.244.0.7:9376]
```

If you don't see those, try restarting `kube-proxy` with the `-v` flag set to 4, and then look at the logs again.

## Edge case: A Pod fails to reach itself via the Service IP

This might sound unlikely, but it does happen and it is supposed to work.

This can happen when the network is not properly configured for "hairpin" traffic, usually when `kube-proxy` is running in `iptables` mode and Pods are connected with bridge network. The Kubelet exposes a `hairpin-mode` flag that allows endpoints of a Service to loadbalance back to themselves if they try to access their own Service VIP. The `hairpin-mode` flag must either be set to `hairpin-veth` or `promiscuous-bridge`.

The common steps to trouble shoot this are as follows:

- Confirm `hairpin-mode` is set to `hairpin-veth` or `promiscuous-bridge`. You should see something like the below. `hairpin-mode` is set to `promiscuous-bridge` in the following example.

```
ps auxw | grep kubelet
```

```
root      3392  1.1  0.8 186804 65208 ?          Sl   00:51  
11:11 /usr/local/bin/kubelet --enable-debugging-  
handlers=true --config=/etc/kubernetes/manifests --allow-  
privileged=True --v=4 --cluster-dns=10.0.0.10 --cluster-  
domain=cluster.local --configure-cbr0=true --cgroup-root=/ --  
system-cgroups=/system --hairpin-mode=promiscuous-bridge --  
runtime-cgroups=/docker-daemon --kubelet-cgroups=/kubelet --  
babysit-daemons=true --max-pods=110 --serialize-image-  
pulls=false --outofdisk-transition-frequency=0
```

- Confirm the effective `hairpin-mode`. To do this, you'll have to look at `kubelet` log. Accessing the logs depends on your Node OS. On some OSes it is a file, such as `/var/log/kubelet.log`, while other OSes use `journalctl` to access logs. Please be noted that the effective hairpin mode may not match `--hairpin-mode` flag due to compatibility. Check if there is any log lines with key word `hairpin` in `kubelet.log`. There should be log lines indicating the effective hairpin mode, like something below.

```
I0629 00:51:43.648698    3252 kubelet.go:380] Hairpin mode  
set to "promiscuous-bridge"
```

- If the effective hairpin mode is `hairpin-veth`, ensure the Kubelet has the permission to operate in `/sys` on node. If everything works properly, you should see something like:

```
for intf in /sys/devices/virtual/net/cbr0/brif/*; do cat $int  
f/hairpin_mode; done
```

```
1  
1  
1  
1
```

- If the effective hairpin mode is `promiscuous-bridge`, ensure `Kubelet` has the permission to manipulate `linux bridge` on node. If `cbr0` bridge is used and configured properly, you should see:

```
ifconfig cbr0 |grep PROMISC
```

```
UP BROADCAST RUNNING PROMISC MULTICAST MTU:1460 Metric:1
```

- Seek help if none of above works out.

## Seek help

If you get this far, something very strange is happening. Your Service is running, has Endpoints, and your Pods are actually serving. You have DNS working, and `kube-proxy` does not seem to be misbehaving. And yet your Service is not working. Please let us know what is going on, so we can help investigate!

Contact us on [Slack](#) or [Forum](#) or [GitHub](#).

## What's next

Visit [troubleshooting document](#) 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](#).

Last modified August 07, 2020 at 4:46 PM PST: [Tune links in tasks section \(1/2\) \(b8541d212\)](#)

[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Running commands in a Pod](#)
- [Setup](#)
- [Does the Service exist?](#)
- [Does the Service work by DNS name?](#)
  - [Does any Service work by DNS name?](#)

- [Does the Service work by IP?](#)
- [Is the Service defined correctly?](#)
- [Does the Service have any Endpoints?](#)
- [Are the Pods working?](#)
- [Is the kube-proxy working?](#)
  - [Is kube-proxy running?](#)
  - [Is kube-proxy proxying?](#)
  - [Edge case: A Pod fails to reach itself via the Service IP](#)
- [Seek help](#)
- [What's next](#)

# **Debugging Kubernetes nodes with crictl**

**FEATURE STATE:** Kubernetes v1.11 [stable]

*crictl is a command-line interface for CRI-compatible container runtimes. You can use it to inspect and debug container runtimes and applications on a Kubernetes node. crictl and its source are hosted in the [cri-tools](#) repository.*

## **Before you begin**

*crictl requires a Linux operating system with a CRI runtime.*

## **Installing crictl**

*You can download a compressed archive `crictl` from the [cri-tools release page](#), for several different architectures. Download the version that corresponds to your version of Kubernetes. Extract it and move it to a location on your system path, such as `/usr/local/bin/`.*

## **General usage**

*The `crictl` command has several subcommands and runtime flags. Use `crictl help` or `crictl <subcommand> help` for more details.*

*crictl connects to `unix:///var/run/dockershim.sock` by default. For other runtimes, you can set the endpoint in multiple different ways:*

- *By setting flags `--runtime-endpoint` and `--image-endpoint`*
- *By setting environment variables `CARRIER_RUNTIME_ENDPOINT` and `IMAGE_SERVICE_ENDPOINT`*
- *By setting the endpoint in the config file `--config=/etc/crictl.yaml`*

*You can also specify timeout values when connecting to the server and enable or disable debugging, by specifying `timeout` or `debug` values in*

the configuration file or using the `--timeout` and `--debug` command-line flags.

To view or edit the current configuration, view or edit the contents of `/etc/crictl.yaml`.

```
cat /etc/crictl.yaml
runtime-endpoint: unix:///var/run/dockershim.sock
image-endpoint: unix:///var/run/dockershim.sock
timeout: 10
debug: true
```

## Example crictl commands

The following examples show some `crictl` commands and example output.

**Warning:** If you use `crictl` to create pod sandboxes or containers on a running Kubernetes cluster, the Kubelet will eventually delete them. `crictl` is not a general purpose workflow tool, but a tool that is useful for debugging.

### List pods

List all pods:

```
crictl pods
```

The output is similar to this:

POD ID	CREATED	NAMESPACE	STATE	ATTEMPT
NAME				
926f1b5a1d33a	About a minute ago	default	Ready	0
sh-84d7dcf559-4r2gq		default		
4dcc216c4adb	About a minute ago	default	Ready	0
nginx-65899c769f-wv2gp		default		
a86316e96fa89	17 hours ago	default	Ready	0
kube-proxy-gblk4		kube-system		
919630b8f81f1	17 hours ago	default	Ready	0
nvidia-device-plugin-zgbvv		kube-system		

List pods by name:

```
crictl pods --name nginx-65899c769f-wv2gp
```

The output is similar to this:

POD ID	CREATED	NAMESPACE	STATE	ATTEMPT
NAME				
4dcc216c4adb	2 minutes ago	default	Ready	0
nginx-65899c769f-wv2gp		default		

*List pods by label:*

```
crictl pods --label run=nginx
```

*The output is similar to this:*

POD ID	CREATED	STATE
NAME	NAMESPACE	ATTEMPT
4dccb216c4adb	2 minutes ago	Ready
nginx-65899c769f-wv2gp	default	0

## **List images**

*List all images:*

```
crictl images
```

*The output is similar to this:*

IMAGE	IMAGE ID	SIZE
busybox		
latest	8c811b4aec35f	1.15MB
k8s-gcrio.azureedge.net/hyperkube-amd64		
v1.10.3	e179bbfe5d238	665MB
k8s-gcrio.azureedge.net/pause-amd64		
3.1	da86e6ba6ca19	742kB
nginx		
latest	cd5239a0906a6	109MB

*List images by repository:*

```
crictl images nginx
```

*The output is similar to this:*

IMAGE	TAG	IMAGE ID
nginx	latest	cd5239a0906a6
109MB		

*Only list image IDs:*

```
crictl images -q
```

*The output is similar to this:*

```
sha256:8c811b4aec35f259572d0f79207bc0678df4c736eec50bc9fec37  
ed936a472a  
sha256:e179bbfe5d238de6069f3b03fccbecc3fb4f2019af741bfff1233c  
4d7b2970c5  
sha256:da86e6ba6ca197bf6bc5e9d900feb906b133eaa4750e6bed647b0  
fbe50ed43e
```

```
sha256:cd5239a0906a6ccf0562354852fae04bc5b52d72a2aff9a871ddb6  
bd57553569
```

## List containers

List all containers:

```
crlctl ps -a
```

The output is similar to this:

CONTAINER ID	IMAGE	CREATED	STATE	ATTEMPT	
1f73f2d81bf98	busybox@sha256:141c253bc4c3fd0a201d32dc1f493bcf3fff003b6df416	dea4f41046e0f37d47	minutes ago	Running	7
sh	9c5951df22c78			1	
87d3992f84f74	busybox@sha256:141c253bc4c3fd0a201d32dc1f493bcf3fff003b6df416	dea4f41046e0f37d47	minutes ago	Exited	8
nginx	nginx@sha256:d0a8828ccb73397acb0073bf34f4d7d8aa315263f1e7806	bf8c55d8ac139d5f		0	
kube-proxy	1941fb4da154f kube-proxy.azureedge.net/hyperkube-amd64@sha256:00d814b1f7763f4ab5be80c58e98140dfc69df107f253d7f	dd714b30a714260a	18 hours ago	Running	6
				0	

List running containers:

```
crlctl ps
```

The output is similar to this:

CONTAINER ID	IMAGE	CREATED	STATE	ATTEMPT	
1f73f2d81bf98	busybox@sha256:141c253bc4c3fd0a201d32dc1f493bcf3fff003b6df416	dea4f41046e0f37d47	minutes ago	Running	6
sh	87d3992f84f74			1	

```
nginx@sha256:d0a8828cccb73397acb0073bf34f4d7d8aa315263f1e7806
bf8c55d8ac139d5f
7
minutes ago      Running
nginx            0
1941fb4da154f   k8s-gcrio.azureedge.net/hyperkube-
amd64@sha256:00d814b1f7763f4ab5be80c58e98140dfc69df107f253d7f
dd714b30a714260a 17 hours ago      Running
kube-proxy        0
```

## Execute a command in a running container

```
cubectl exec -i -t 1f73f2d81bf98 ls
```

The output is similar to this:

```
bin  dev  etc  home  proc  root  sys  tmp  usr  var
```

## Get a container's logs

Get all container logs:

```
cubectl logs 87d3992f84f74
```

The output is similar to this:

```
10.240.0.96 - - [06/Jun/2018:02:45:49 +0000] "GET / HTTP/
1.1" 200 612 "-" "curl/7.47.0" "-"
10.240.0.96 - - [06/Jun/2018:02:45:50 +0000] "GET / HTTP/
1.1" 200 612 "-" "curl/7.47.0" "-"
10.240.0.96 - - [06/Jun/2018:02:45:51 +0000] "GET / HTTP/
1.1" 200 612 "-" "curl/7.47.0" "-"
```

Get only the latest N lines of logs:

```
cubectl logs --tail=1 87d3992f84f74
```

The output is similar to this:

```
10.240.0.96 - - [06/Jun/2018:02:45:51 +0000] "GET / HTTP/
1.1" 200 612 "-" "curl/7.47.0" "-"
```

## Run a pod sandbox

Using `crictl` to run a pod sandbox is useful for debugging container runtimes. On a running Kubernetes cluster, the sandbox will eventually be stopped and deleted by the Kubelet.

1. Create a JSON file like the following:

```
{
  "metadata": {
    "name": "nginx-sandbox",
    "namespace": "default",
```

```
        "attempt": 1,
        "uid": "hdishd83djaidwnduwk28bcsb"
    },
    "logDirectory": "/tmp",
    "linux": {
    }
}
```

2. Use the `cricctl runp` command to apply the JSON and run the sandbox.

```
cricctl runp pod-config.json
```

The ID of the sandbox is returned.

## Create a container

Using `cricctl` to create a container is useful for debugging container runtimes. On a running Kubernetes cluster, the sandbox will eventually be stopped and deleted by the Kubelet.

1. Pull a busybox image

```
cricctl pull busybox
Image is up to date for
busybox@sha256:141c253bc4c3fd0a201d32dc1f493bcf3fff003b6
df416dea4f41046e0f37d47
```

2. Create configs for the pod and the container:

### Pod config:

```
{
  "metadata": {
    "name": "nginx-sandbox",
    "namespace": "default",
    "attempt": 1,
    "uid": "hdishd83djaidwnduwk28bcsb"
  },
  "log_directory": "/tmp",
  "linux": {
  }
}
```

### Container config:

```
{
  "metadata": {
    "name": "busybox"
  },
  "image": {
    "image": "busybox"
  },
}
```

```

"command": [
    "top"
],
"log_path": "busybox.log",
"linux": {
}
}

```

3. Create the container, passing the ID of the previously-created pod, the container config file, and the pod config file. The ID of the container is returned.

```

crictl create
f84dd361f8dc51518ed291fbadd6db537b0496536c1d2d6c05ff943c
e8c9a54f container-config.json pod-config.json

```

4. List all containers and verify that the newly-created container has its state set to *Created*.

```
crictl ps -a
```

The output is similar to this:

<i>CONTAINER ID</i>	<i>IMAGE</i>
<i>CREATED</i>	<i>STATE</i>
<i>NAME</i>	<i>ATTEMPT</i>
3e025dd50a72d ago	busybox <i>busybox</i>
	32 seconds 0

## **Start a container**

To start a container, pass its ID to *crictl start*:

```

crictl start
3e025dd50a72d956c4f14881fbb5b1080c9275674e95fb67f965f6478a957
d60

```

The output is similar to this:

```
3e025dd50a72d956c4f14881fbb5b1080c9275674e95fb67f965f6478a957
d60
```

Check the container has its state set to *Running*.

```
crictl ps
```

The output is similar to this:

<i>CONTAINER ID</i>	<i>IMAGE</i>	<i>CREATED</i>
<i>STATE</i>	<i>NAME</i>	<i>ATTEMPT</i>
3e025dd50a72d Running	busybox busybox	About a minute ago 0

See [kubernetes-sigs/cri-tools](#) for more information.

## Mapping from docker cli to crictl

The exact versions for below mapping table are for docker cli v1.40 and crictl v1.19.0. Please note that the list is not exhaustive. For example, it doesn't include experimental commands of docker cli.

**Note:** Warn: the output format of CRICTL is similar to Docker CLI, despite some missing columns for some CLI. Make sure to check output for the specific command if your script output parsing.

### Retrieve Debugging Information

<b>docker cli</b>	<b>crictl</b>	<b>Description</b>	<b>Unsupported Features</b>
<code>attach</code>	<code>attach</code>	Attach to a running container	<code>--detach-keys</code> , <code>--sig-proxy</code>
<code>exec</code>	<code>exec</code>	Run a command in a running container	<code>--privileged</code> , <code>--user</code> , <code>--detach-keys</code>
<code>images</code>	<code>images</code>	List images	Â
<code>info</code>	<code>info</code>	Display system-wide information	Â
<code>inspect</code>	<code>inspect</code> , <code>inspect i</code>	Return low-level information on a container, image or task	Â
<code>logs</code>	<code>logs</code>	Fetch the logs of a container	<code>--details</code>
<code>ps</code>	<code>ps</code>	List containers	Â
<code>stats</code>	<code>stats</code>	Display a live stream of container(s) resource usage statistics	Column: NET/BLOCK I/O, PIDs
<code>version</code>	<code>version</code>	Show the runtime (Docker, ContainerD, or others) version information	Â

### Perform Changes

<b>docker cli</b>	<b>crictl</b>	<b>Description</b>	<b>Unsupported Features</b>
<code>create</code>	<code>create</code>	Create a new container	Â
<code>kill</code>	<code>stop (timeout = 0)</code>	Kill one or more running container	<code>--signal</code>

<b>docker cli</b>	<b>cricctl</b>	<b>Description</b>	<b>Unsupported Features</b>
<i>pull</i>	<i>pull</i>	<i>Pull an image or a repository from a registry</i>	<i>--all-tags, --disable-content-trust</i>
<i>rm</i>	<i>rm</i>	<i>Remove one or more containers</i>	Ã
<i>rmi</i>	<i>rmi</i>	<i>Remove one or more images</i>	Ã
<i>run</i>	<i>run</i>	<i>Run a command in a new container</i>	Ã
<i>start</i>	<i>start</i>	<i>Start one or more stopped containers</i>	<i>--detach-keys</i>
<i>stop</i>	<i>stop</i>	<i>Stop one or more running containers</i>	Ã
<i>update</i>	<i>update</i>	<i>Update configuration of one or more containers</i>	<i>--restart, --blkio-weight and some other resource limit not supported by CRI.</i>

## **Supported only in cricctl**

<b>cricctl</b>	<b>Description</b>
<i>imagefsinfo</i>	<i>Return image filesystem info</i>
<i>inspectp</i>	<i>Display the status of one or more pods</i>
<i>port-forward</i>	<i>Forward local port to a pod</i>
<i>pods</i>	<i>List pods</i>
<i>runp</i>	<i>Run a new pod</i>
<i>rmp</i>	<i>Remove one or more pods</i>
<i>stopp</i>	<i>Stop one or more running pods</i>

## **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](#).

Last modified December 08, 2020 at 3:49 PM PST: [add note shortcode to 'Debugging Kubernetes nodes with cricctl'](#) (`c86ca1a57`)

[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Installing cricctl](#)
- [General usage](#)
- [Example cricctl commands](#)
  - [List pods](#)

- [List images](#)
- [List containers](#)
- [Execute a command in a running container](#)
- [Get a container's logs](#)
- [Run a pod sandbox](#)
- [Create a container](#)
- [Start a container](#)
- [Mapping from docker cli to cri<code>t</code>l](#)
  - [Retrieve Debugging Information](#)
  - [Perform Changes](#)
  - [Supported only in cri<code>t</code>l](#)

# **Determine the Reason for Pod Failure**

*This page shows how to write and read a Container termination message.*

*Termination messages provide a way for containers to write information about fatal events to a location where it can be easily retrieved and surfaced by tools like dashboards and monitoring software. In most cases, information that you put in a termination message should also be written to the general [Kubernetes logs](#).*

## **Before you begin**

*You need to have a Kubernetes cluster, and the kubectl command-line tool must be configured to communicate with your cluster. If you do not already have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:*

- [Katacoda](#)
- [Play with Kubernetes](#)

*To check the version, enter `kubectl version`.*

## **Writing and reading a termination message**

*In this exercise, you create a Pod that runs one container. The configuration file specifies a command that runs when the container starts.*

[debug/termination.yaml](#)



```
apiVersion: v1
kind: Pod
```

```
metadata:
  name: termination-demo
spec:
  containers:
    - name: termination-demo-container
      image: debian
      command: ["/bin/sh"]
      args: ["-c", "sleep 10 && echo Sleep expired > /dev/termination-log"]
```

1. Create a Pod based on the YAML configuration file:

```
kubectl apply -f https://k8s.io/examples/debug/termination.yaml
```

In the YAML file, in the `cmd` and `args` fields, you can see that the container sleeps for 10 seconds and then writes "Sleep expired" to the `/dev/termination-log` file. After the container writes the "Sleep expired" message, it terminates.

2. Display information about the Pod:

```
kubectl get pod termination-demo
```

Repeat the preceding command until the Pod is no longer running.

3. Display detailed information about the Pod:

```
kubectl get pod termination-demo --output=yaml
```

The output includes the "Sleep expired" message:

```
apiVersion: v1
kind: Pod
...
  lastState:
    terminated:
      containerID: ...
      exitCode: 0
      finishedAt: ...
      message: |
        Sleep expired
...
```

4. Use a Go template to filter the output so that it includes only the termination message:

```
kubectl get pod termination-demo -o go-template="{{range .status.containerStatuses}}{{.lastState.terminated.message}}{{end}}"
```

# Customizing the termination message

Kubernetes retrieves termination messages from the termination message file specified in the `terminationMessagePath` field of a Container, which has a default value of `/dev/termination-log`. By customizing this field, you can tell Kubernetes to use a different file. Kubernetes use the contents from the specified file to populate the Container's status message on both success and failure.

The termination message is intended to be brief final status, such as an assertion failure message. The kubelet truncates messages that are longer than 4096 bytes. The total message length across all containers will be limited to 12KiB. The default termination message path is `/dev/termination-log`. You cannot set the termination message path after a Pod is launched

In the following example, the container writes termination messages to `/tmp/my-log` for Kubernetes to retrieve:

```
apiVersion: v1
kind: Pod
metadata:
  name: msg-path-demo
spec:
  containers:
    - name: msg-path-demo-container
      image: debian
      terminationMessagePath: "/tmp/my-log"
```

Moreover, users can set the `terminationMessagePolicy` field of a Container for further customization. This field defaults to "File" which means the termination messages are retrieved only from the termination message file. By setting the `terminationMessagePolicy` to "FallbackToLogsOnError", you can tell Kubernetes to use the last chunk of container log output if the termination message file is empty and the container exited with an error. The log output is limited to 2048 bytes or 80 lines, whichever is smaller.

## What's next

- See the `terminationMessagePath` field in [Container](#).
- Learn about [retrieving logs](#).
- Learn about [Go templates](#).

# **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](#).

Last modified July 04, 2020 at 2:07 PM PST: [Corrected a Typo](#) ([e6fd8f866](#))

[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Writing and reading a termination message](#)
- [Customizing the termination message](#)
- [What's next](#)

# **Developing and debugging services locally**

Kubernetes applications usually consist of multiple, separate services, each running in its own container. Developing and debugging these services on a remote Kubernetes cluster can be cumbersome, requiring you to [get a shell on a running container](#) and running your tools inside the remote shell.

`telepresence` is a tool to ease the process of developing and debugging services locally, while proxying the service to a remote Kubernetes cluster. Using `telepresence` allows you to use custom tools, such as a debugger and IDE, for a local service and provides the service full access to ConfigMap, secrets, and the services running on the remote cluster.

This document describes using `telepresence` to develop and debug services running on a remote cluster locally.

## **Before you begin**

- *Kubernetes cluster is installed*
- *kubectl is configured to communicate with the cluster*
- [Telepresence](#) is installed

## **Getting a shell on a remote cluster**

*Open a terminal and run `telepresence` with no arguments to get a `telepresence` shell. This shell runs locally, giving you full access to your local filesystem.*

*The `telepresence` shell can be used in a variety of ways. For example, write a shell script on your laptop, and run it directly from the shell in real time. You can do this on a remote shell as well, but you might not be able to use your preferred code editor, and the script is deleted when the container is terminated.*

*Enter `exit` to quit and close the shell.*

## **Developing or debugging an existing service**

*When developing an application on Kubernetes, you typically program or debug a single service. The service might require access to other services for testing and debugging. One option is to use the continuous deployment pipeline, but even the fastest deployment pipeline introduces a delay in the program or debug cycle.*

*Use the `--swap-deployment` option to swap an existing deployment with the Telepresence proxy. Swapping allows you to run a service locally and connect to the remote Kubernetes cluster. The services in the remote cluster can now access the locally running instance.*

*To run `telepresence` with `--swap-deployment`, enter:*

```
telepresence --swap-deployment $DEPLOYMENT_NAME
```

*where `$DEPLOYMENT_NAME` is the name of your existing deployment.*

*Running this command spawns a shell. In the shell, start your service. You can then make edits to the source code locally, save, and see the changes take effect immediately. You can also run your service in a debugger, or any other local development tool.*

## **What's next**

*If you're interested in a hands-on tutorial, check out [this tutorial](#) that walks through locally developing the Guestbook application on Google Kubernetes Engine.*

*Telepresence has [numerous proxying options](#), depending on your situation.*

*For further reading, visit the [Telepresence website](#).*

# 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](#).

Last modified May 30, 2020 at 3:10 PM PST: [add en pages \(ecc27bbbe\)](#)  
[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Getting a shell on a remote cluster](#)
- [Developing or debugging an existing service](#)
- [What's next](#)

# Events in Stackdriver

Kubernetes events are objects that provide insight into what is happening inside a cluster, such as what decisions were made by scheduler or why some pods were evicted from the node. You can read more about using events for debugging your application in the [Application Introspection and Debugging](#) section.

Since events are API objects, they are stored in the apiserver on master. To avoid filling up master's disk, a retention policy is enforced: events are removed one hour after the last occurrence. To provide longer history and aggregation capabilities, a third party solution should be installed to capture events.

This article describes a solution that exports Kubernetes events to Stackdriver Logging, where they can be processed and analyzed.

**Note:** It is not guaranteed that all events happening in a cluster will be exported to Stackdriver. One possible scenario when events will not be exported is when event exporter is not running (e.g. during restart or upgrade). In most cases it's fine to use events for purposes like setting up [metrics](#) and [alerts](#), but you should be aware of the potential inaccuracy.

# Deployment

## Google Kubernetes Engine

In Google Kubernetes Engine, if cloud logging is enabled, event exporter is deployed by default to the clusters with master running version 1.7 and higher. To prevent disturbing your workloads, event exporter does not have resources set and is in the best effort QOS

*class, which means that it will be the first to be killed in the case of resource starvation. If you want your events to be exported, make sure you have enough resources to facilitate the event exporter pod. This may vary depending on the workload, but on average, approximately 100Mb RAM and 100m CPU is needed.*

## **Deploying to the Existing Cluster**

*Deploy event exporter to your cluster using the following command:*

```
kubectl apply -f https://k8s.io/examples/debug/event-exporter.yaml
```

*Since event exporter accesses the Kubernetes API, it requires permissions to do so. The following deployment is configured to work with RBAC authorization. It sets up a service account and a cluster role binding to allow event exporter to read events. To make sure that event exporter pod will not be evicted from the node, you can additionally set up resource requests. As mentioned earlier, 100Mb RAM and 100m CPU should be enough.*

[debug/event-exporter.yaml](#)



```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: event-exporter-sa
  namespace: default
  labels:
    app: event-exporter
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: event-exporter-rb
  labels:
    app: event-exporter
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: view
subjects:
- kind: ServiceAccount
  name: event-exporter-sa
  namespace: default
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: event-exporter-v0.2.3
  namespace: default
```

```
labels:
  app: event-exporter
spec:
  selector:
    matchLabels:
      app: event-exporter
  replicas: 1
  template:
    metadata:
      labels:
        app: event-exporter
    spec:
      serviceAccountName: event-exporter-sa
      containers:
        - name: event-exporter
          image: k8s.gcr.io/event-exporter:v0.2.3
          command:
            - '/event-exporter'
      terminationGracePeriodSeconds: 30
```

## User Guide

*Events are exported to the GKE Cluster resource in Stackdriver Logging. You can find them by selecting an appropriate option from a drop-down menu of available resources:*

Filter by label or text search

GKE Cluster      All logs      Any log level

BigQuery	
GCE Disk	porter started watching. Some events may have been
GCE Firewall Rule	replica set nginx-deployment-3088474477 to 2
GCE Instance Group Manager	pod: nginx-deployment-3088474477-w3hzb
GCE Instance Template	pod: nginx-deployment-3088474477-9bbgf
GCE Network	lly assigned nginx-deployment-3088474477-9bbgf
GCE Project	lly assigned nginx-deployment-3088474477-w3hzb
GCE Reserved Address	ne.SetUp succeeded for volume "default-token-m1"
GCE Route	ne.SetUp succeeded for volume "default-token-m1"
GCE VM Instance	nage "nginx:1.7.9"
<input checked="" type="checkbox"/> GKE Cluster	nage "nginx:1.7.9"
GKE Container	lly pulled image "nginx:1.7.9"
Global	lly pulled image "nginx:1.7.9"
Google Project	ontainer
Logging export sink	ontainer
Service Account	ontainer

19:38:50.000 Started container

No newer entries found matching current filter

You can filter based on the event object fields using Stackdriver Logging [filtering mechanism](#). For example, the following query will show events from the scheduler about pods from deployment nginx-deployment:

```
resource.type="gke_cluster"
jsonPayload.kind="Event"
jsonPayload.source.component="default-scheduler"
jsonPayload.involvedObject.name:"nginx-deployment"
```

```
1 resource.type="gke_cluster"
2 jsonPayload.kind="Event"
3 jsonPayload.source.component="default-scheduler"
4 jsonPayload.involvedObject.name:"nginx-deployment"
```

Submit Filter    Jump to date ▾

2017-06-21 CEST

↓

▶ i 19:38:41.000 Successfully assigned nginx-deployment-3088474477-9bbgf

▶ i 19:38:41.000 Successfully assigned nginx-deployment-3088474477-w3hzb

↑

## 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](#).

Last modified May 30, 2020 at 3:10 PM PST: [add en pages \(ecc27bbbe\)](#)  
[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Deployment](#)
  - [Google Kubernetes Engine](#)
  - [Deploying to the Existing Cluster](#)
- [User Guide](#)

## Get a Shell to a Running Container

This page shows how to use `kubectl exec` to get a shell to a running container.

### Before you begin

You need to have a Kubernetes cluster, and the `kubectl` command-line tool must be configured to communicate with your cluster. If you do not already have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:

- [Katacoda](#)
- [Play with Kubernetes](#)

# Getting a shell to a container

In this exercise, you create a Pod that has one container. The container runs the nginx image. Here is the configuration file for the Pod:

[application/shell-demo.yaml](#)



```
apiVersion: v1
kind: Pod
metadata:
  name: shell-demo
spec:
  volumes:
    - name: shared-data
      emptyDir: {}
  containers:
    - name: nginx
      image: nginx
      volumeMounts:
        - name: shared-data
          mountPath: /usr/share/nginx/html
  hostNetwork: true
  dnsPolicy: Default
```

Create the Pod:

```
kubectl apply -f https://k8s.io/examples/application/shell-
demo.yaml
```

Verify that the container is running:

```
kubectl get pod shell-demo
```

Get a shell to the running container:

```
kubectl exec --stdin --tty shell-demo -- /bin/bash
```

**Note:** The double dash (--) separates the arguments you want to pass to the command from the kubectl arguments.

In your shell, list the root directory:

```
# Run this inside the container
ls /
```

In your shell, experiment with other commands. Here are some examples:

```
# You can run these example commands inside the container
ls /
cat /proc/mounts
```

```
cat /proc/1/maps
apt-get update
apt-get install -y tcpdump
tcpdump
apt-get install -y lsof
lsof
apt-get install -y procps
ps aux
ps aux | grep nginx
```

## **Writing the root page for nginx**

*Look again at the configuration file for your Pod. The Pod has an empty Dir volume, and the container mounts the volume at /usr/share/nginx/html.*

*In your shell, create an index.html file in the /usr/share/nginx/html directory:*

```
# Run this inside the container
echo 'Hello shell demo' > /usr/share/nginx/html/index.html
```

*In your shell, send a GET request to the nginx server:*

```
# Run this in the shell inside your container
apt-get update
apt-get install curl
curl http://localhost/
```

*The output shows the text that you wrote to the index.html file:*

```
Hello shell demo
```

*When you are finished with your shell, enter exit.*

```
exit # To quit the shell in the container
```

## **Running individual commands in a container**

*In an ordinary command window, not your shell, list the environment variables in the running container:*

```
kubectl exec shell-demo env
```

*Experiment with running other commands. Here are some examples:*

```
kubectl exec shell-demo -- ps aux
kubectl exec shell-demo -- ls /
kubectl exec shell-demo -- cat /proc/1/mounts
```

# **Opening a shell when a Pod has more than one container**

If a Pod has more than one container, use `--container` or `-c` to specify a container in the `kubectl exec` command. For example, suppose you have a Pod named `my-pod`, and the Pod has two containers named `main-app` and `helper-app`. The following command would open a shell to the `main-app` container.

```
kubectl exec -i -t my-pod --container main-app -- /bin/bash
```

**Note:** The short options `-i` and `-t` are the same as the long options `--stdin` and `--tty`

## **What's next**

- Read about [kubectl exec](#)

## **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](#).

Last modified May 23, 2020 at 6:18 PM PST: [Revise sample commands to match style guide \(280a527a7\)](#)

[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Getting a shell to a container](#)
- [Writing the root page for nginx](#)
- [Running individual commands in a container](#)
- [Opening a shell when a Pod has more than one container](#)
- [What's next](#)

# **Logging Using Elasticsearch and Kibana**

On the Google Compute Engine (GCE) platform, the default logging support targets [Stackdriver Logging](#), which is described in detail in the [Logging With Stackdriver Logging](#).

This article describes how to set up a cluster to ingest logs into [Elasticsearch](#) and view them using [Kibana](#), as an alternative to Stackdriver Logging when running on GCE.

**Note:** You cannot automatically deploy Elasticsearch and Kibana in the Kubernetes cluster hosted on Google Kubernetes Engine. You have to deploy them manually.

To use Elasticsearch and Kibana for cluster logging, you should set the following environment variable as shown below when creating your cluster with `kube-up.sh`:

```
KUBE_LOGGING_DESTINATION=elasticsearch
```

You should also ensure that `KUBE_ENABLE_NODE_LOGGING=true` (which is the default for the GCE platform).

Now, when you create a cluster, a message will indicate that the Fluentd log collection daemons that run on each node will target Elasticsearch:

```
cluster/kube-up.sh
```

```
...
Project: kubernetes-satnam
Zone: us-central1-b
... calling kube-up
Project: kubernetes-satnam
Zone: us-central1-b
+++ Staging server tars to Google Storage: gs://kubernetes-
staging-e6d0e81793/devel
+++ kubernetes-server-linux-amd64.tar.gz uploaded (sha1 =
6987c098277871b6d69623141276924ab687f89d)
+++ kubernetes-salt.tar.gz uploaded (sha1 =
bdfc83ed6b60fa9e3bff9004b542cfcc643464cd0)
Looking for already existing resources
Starting master and configuring firewalls
Created [https://www.googleapis.com/compute/v1/projects/
kubernetes-satnam/zones/us-central1-b/disks/kubernetes-
master-pd].
NAME          ZONE      SIZE_GB  TYPE    STATUS
kubernetes-master-pd  us-central1-b  20      pd-ssd  READY
Created [https://www.googleapis.com/compute/v1/projects/
kubernetes-satnam/regions/us-central1/addresses/kubernetes-
master-ip].
+++ Logging using Fluentd to elasticsearch
```

The per-node Fluentd pods, the Elasticsearch pods, and the Kibana pods should all be running in the `kube-system` namespace soon after the cluster comes to life.

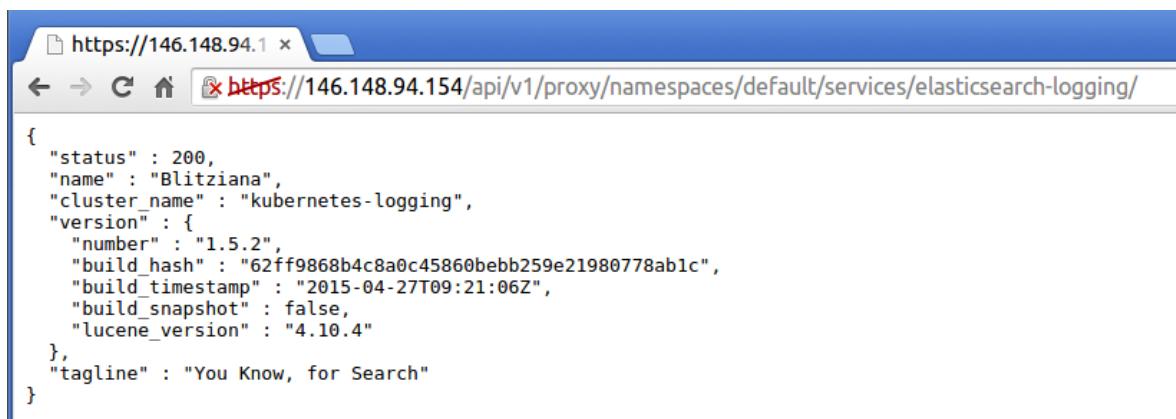
```
kubectl get pods --namespace=kube-system
```

NAME			
STATUS	RESTARTS	AGE	
elasticsearch-logging-v1-78nog		2h	1/1
Running	0	2h	
elasticsearch-logging-v1-nj2nb		2h	1/1
Running	0	2h	
fluentd-elasticsearch-kubernetes-node-5oq0		2h	1/1
Running	0	2h	
fluentd-elasticsearch-kubernetes-node-6896		2h	1/1
Running	0	2h	
fluentd-elasticsearch-kubernetes-node-l1ds		2h	1/1
Running	0	2h	
fluentd-elasticsearch-kubernetes-node-lz9j		2h	1/1
Running	0	2h	
kibana-logging-v1-bhp08		2h	1/1
Running	0	2h	
kube-dns-v3-7r1l9			3/3
Running	0	2h	
monitoring-heapster-v4-yl332		2h	1/1
Running	1	2h	
monitoring-influx-grafana-v1-o79xf			2/2
Running	0	2h	

The `fluentd-elasticsearch` pods gather logs from each node and send them to the `elasticsearch-logging` pods, which are part of a [service](#) named `elasticsearch-logging`. These Elasticsearch pods store the logs and expose them via a REST API. The `kibana-logging` pod provides a web UI for reading the logs stored in Elasticsearch, and is part of a service named `kibana-logging`.

The Elasticsearch and Kibana services are both in the `kube-system` namespace and are not directly exposed via a publicly reachable IP address. To reach them, follow the instructions for [Accessing services running in a cluster](#).

If you try accessing the `elasticsearch-logging` service in your browser, you'll see a status page that looks something like this:



```

https://146.148.94.154/api/v1/proxy/namespaces/default/services/elasticsearch-logging/_status

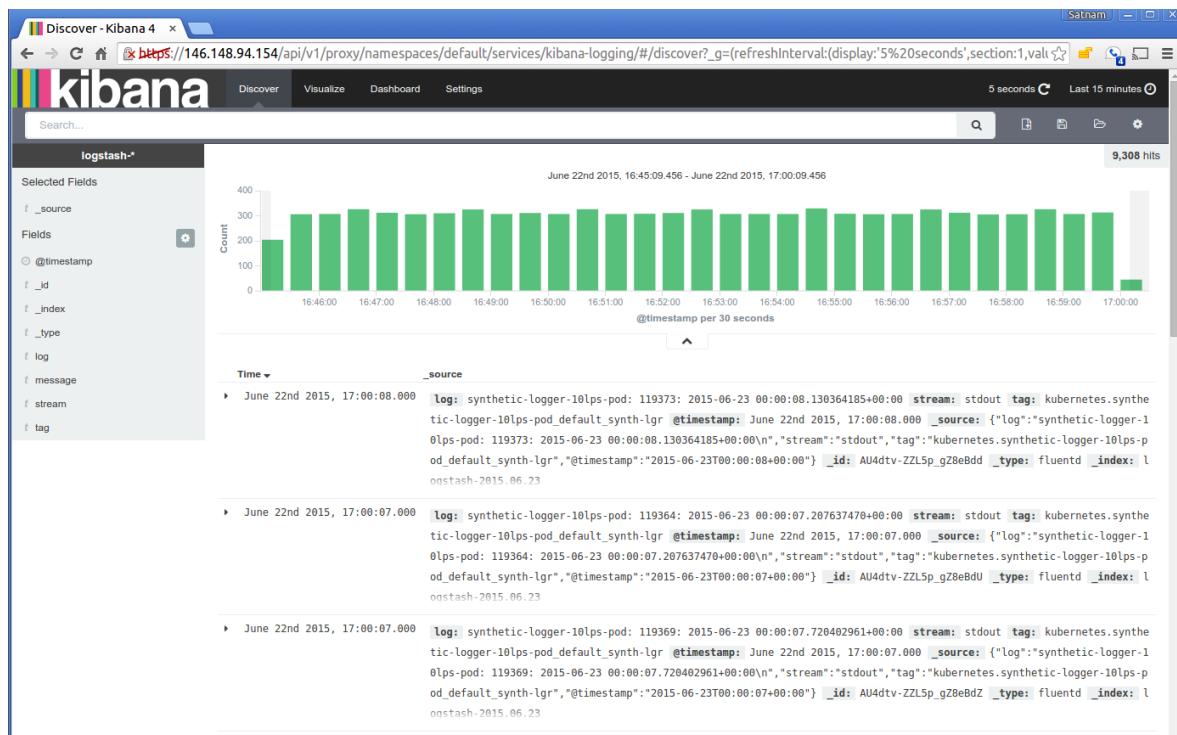
{
  "status" : 200,
  "name" : "Blitziana",
  "cluster_name" : "kubernetes-logging",
  "version" : {
    "number" : "1.5.2",
    "build_hash" : "62ff9868b4c8a0c45860bebb259e21980778ab1c",
    "build_timestamp" : "2015-04-27T09:21:06Z",
    "build_snapshot" : false,
    "lucene_version" : "4.10.4"
  },
  "tagline" : "You Know, for Search"
}

```

You can now type Elasticsearch queries directly into the browser, if you'd like. See [Elasticsearch's documentation](#) for more details on how to do so.

Alternatively, you can view your cluster's logs using Kibana (again using the [instructions for accessing a service running in the cluster](#)). The first time you visit the Kibana URL you will be presented with a page that asks you to configure your view of the ingested logs. Select the option for timeseries values and select `@timestamp`. On the following page select the `Discover` tab and then you should be able to see the ingested logs. You can set the refresh interval to 5 seconds to have the logs regularly refreshed.

Here is a typical view of ingested logs from the Kibana viewer:



## What's next

Kibana opens up all sorts of powerful options for exploring your logs! For some ideas on how to dig into it, check out [Kibana's documentation](#).

## 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](#).

Last modified August 07, 2020 at 4:46 PM PST: [Tune links in tasks section \(1/2\) \(b8541d212\)](#)

[Edit this page](#) [Create child page](#) [Create an issue](#)

- [What's next](#)

# Logging Using Stackdriver

Before reading this page, it's highly recommended to familiarize yourself with the [overview of logging in Kubernetes](#).

**Note:** By default, Stackdriver logging collects only your container's standard output and standard error streams. To collect any logs your application writes to a file (for example), see the [sidecar approach](#) in the Kubernetes logging overview.

## Deploying

To ingest logs, you must deploy the Stackdriver Logging agent to each node in your cluster. The agent is a configured fluentd instance, where the configuration is stored in a ConfigMap and the instances are managed using a Kubernetes DaemonSet. The actual deployment of the ConfigMap and DaemonSet for your cluster depends on your individual cluster setup.

### Deploying to a new cluster

#### Google Kubernetes Engine

Stackdriver is the default logging solution for clusters deployed on Google Kubernetes Engine. Stackdriver Logging is deployed to a new cluster by default unless you explicitly opt-out.

#### Other platforms

To deploy Stackdriver Logging on a new cluster that you're creating using `kube-up.sh`, do the following:

1. Set the `KUBE_LOGGING_DESTINATION` environment variable to `gcp`.
2. **If not running on GCE**, include the `beta.kubernetes.io/fluentd-ds-ready=true` in the `KUBE_NODE_LABELS` variable.

Once your cluster has started, each node should be running the Stackdriver Logging agent. The DaemonSet and ConfigMap are configured as addons. If you're not using `kube-up.sh`, consider starting a cluster without a pre-configured logging solution and then deploying Stackdriver Logging agents to the running cluster.

**Warning:** The Stackdriver logging daemon has known issues on platforms other than Google Kubernetes Engine. Proceed at your own risk.

## **Deploying to an existing cluster**

1. Apply a label on each node, if not already present.

*The Stackdriver Logging agent deployment uses node labels to determine to which nodes it should be allocated. These labels were introduced to distinguish nodes with the Kubernetes version 1.6 or higher. If the cluster was created with Stackdriver Logging configured and node has version 1.5.X or lower, it will have fluentd as static pod. Node cannot have more than one instance of fluentd, therefore only apply labels to the nodes that don't have fluentd pod allocated already. You can ensure that your node is labelled properly by running kubectl describe as follows:*

```
kubectl describe node $NODE_NAME
```

*The output should be similar to this:*

Name:	NODE_NAME
Role:	
Labels:	beta.kubernetes.io/fluentd-ds-ready=true
...	

*Ensure that the output contains the label beta.kubernetes.io/fluentd-ds-ready=true. If it is not present, you can add it using the kubectl label command as follows:*

```
kubectl label node $NODE_NAME beta.kubernetes.io/fluentd-ds-ready=true
```

**Note:** If a node fails and has to be recreated, you must re-apply the label to the recreated node. To make this easier, you can use Kubelet's command-line parameter for applying node labels in your node startup script.

2. Deploy a ConfigMap with the logging agent configuration by running the following command:

```
kubectl apply -f https://k8s.io/examples/debug/fluentd-gcp-configmap.yaml
```

*The command creates the ConfigMap in the default namespace. You can download the file manually and change it before creating the ConfigMap object.*

3. Deploy the logging agent DaemonSet by running the following command:

```
kubectl apply -f https://k8s.io/examples/debug/fluentd-gcp-ds.yaml
```

*You can download and edit this file before using it as well.*

# Verifying your Logging Agent Deployment

After Stackdriver DaemonSet is deployed, you can discover logging agent deployment status by running the following command:

```
kubectl get ds --all-namespaces
```

If you have 3 nodes in the cluster, the output should looks similar to this:

NAMESPACE READY	NAME NODE-SELECTOR	DESIRED	CURRENT	AGE
...				
default 3	fluentd-gcp-v2.0 beta.kubernetes.io/fluentd-ds-ready=true	3	3	5m
...				

To understand how logging with Stackdriver works, consider the following synthetic log generator pod specification [counter-pod.yaml](#):

[debug/counter-pod.yaml](#)



```
apiVersion: v1
kind: Pod
metadata:
  name: counter
spec:
  containers:
    - name: count
      image: busybox
      args: [/bin/sh, -c,
              'i=0; while true; do echo "$i: $(date)"; i=$((i+1)); sleep 1; done']
```

This pod specification has one container that runs a bash script that writes out the value of a counter and the datetime once per second, and runs indefinitely. Let's create this pod in the default namespace.

```
kubectl apply -f https://k8s.io/examples/debug/counter-pod.yaml
```

You can observe the running pod:

```
kubectl get pods
```

NAME			
STATUS	RESTARTS	AGE	READY
counter			1/1
Running	0	5m	

*For a short period of time you can observe the 'Pending' pod status, because the kubelet has to download the container image first. When the pod status changes to Running you can use the kubectl logs command to view the output of this counter pod.*

```
kubectl logs counter
```

```
0: Mon Jan  1 00:00:00 UTC 2001
1: Mon Jan  1 00:00:01 UTC 2001
2: Mon Jan  1 00:00:02 UTC 2001
...
```

*As described in the logging overview, this command fetches log entries from the container log file. If the container is killed and then restarted by Kubernetes, you can still access logs from the previous container. However, if the pod is evicted from the node, log files are lost. Let's demonstrate this by deleting the currently running counter container:*

```
kubectl delete pod counter
```

```
pod "counter" deleted
```

*and then recreating it:*

```
kubectl create -f https://k8s.io/examples/debug/counter-
pod.yaml
```

```
pod/counter created
```

*After some time, you can access logs from the counter pod again:*

```
kubectl logs counter
```

```
0: Mon Jan  1 00:01:00 UTC 2001
1: Mon Jan  1 00:01:01 UTC 2001
2: Mon Jan  1 00:01:02 UTC 2001
...
```

*As expected, only recent log lines are present. However, for a real-world application you will likely want to be able to access logs from all containers, especially for the debug purposes. This is exactly when the previously enabled Stackdriver Logging can help.*

## **Viewing logs**

*Stackdriver Logging agent attaches metadata to each log entry, for you to use later in queries to select only the messages you're interested in: for example, the messages from a particular pod.*

*The most important pieces of metadata are the resource type and log name. The resource type of a container log is container, which is named GKE Containers in the UI (even if the Kubernetes cluster is not on Google Kubernetes Engine). The log name is the name of the*

*container, so that if you have a pod with two containers, named `container_1` and `container_2` in the spec, their logs will have log names `container_1` and `container_2` respectively.*

*System components have resource type `compute`, which is named `GCE VM Instance` in the interface. Log names for system components are fixed. For a Google Kubernetes Engine node, every log entry from a system component has one of the following log names:*

- `docker`
- `kubelet`
- `kube-proxy`

*You can learn more about viewing logs on [the dedicated Stackdriver page](#).*

*One of the possible ways to view logs is using the `gcloud logging` command line interface from the [Google Cloud SDK](#). It uses Stackdriver Logging [filtering syntax](#) to query specific logs. For example, you can run the following command:*

```
gcloud beta logging read 'logName="projects/$YOUR_PROJECT_ID/logs/count"' --format json | jq '.[].textPayload'
```

```
...
"2: Mon Jan  1 00:01:02 UTC 2001\n"
"1: Mon Jan  1 00:01:01 UTC 2001\n"
"0: Mon Jan  1 00:01:00 UTC 2001\n"

...
"2: Mon Jan  1 00:00:02 UTC 2001\n"
"1: Mon Jan  1 00:00:01 UTC 2001\n"
"0: Mon Jan  1 00:00:00 UTC 2001\n"
```

*As you can see, it outputs messages for the `count` container from both the first and second runs, despite the fact that the `kubelet` already deleted the logs for the first container.*

## **Exporting logs**

*You can export logs to [Google Cloud Storage](#) or to [BigQuery](#) to run further analysis. Stackdriver Logging offers the concept of sinks, where you can specify the destination of log entries. More information is available on the Stackdriver [Exporting Logs page](#).*

## **Configuring Stackdriver Logging Agents**

*Sometimes the default installation of Stackdriver Logging may not suit your needs, for example:*

- *You may want to add more resources because default performance doesn't suit your needs.*

- You may want to introduce additional parsing to extract more metadata from your log messages, like severity or source code reference.
- You may want to send logs not only to Stackdriver or send it to Stackdriver only partially.

*In this case you need to be able to change the parameters of DaemonSet and ConfigMap.*

## **Prerequisites**

*If you're using GKE and Stackdriver Logging is enabled in your cluster, you cannot change its configuration, because it's managed and supported by GKE. However, you can disable the default integration and deploy your own.*

**Note:** You will have to support and maintain a newly deployed configuration yourself: update the image and configuration, adjust the resources and so on.

*To disable the default logging integration, use the following command:*

```
gcloud beta container clusters update --logging-service=none
CLUSTER
```

*You can find notes on how to then install Stackdriver Logging agents into a running cluster in the [Deploying section](#).*

## **Changing DaemonSet parameters**

*When you have the Stackdriver Logging DaemonSet in your cluster, you can just modify the template field in its spec, daemonset controller will update the pods for you. For example, let's assume you've just installed the Stackdriver Logging as described above. Now you want to change the memory limit to give fluentd more memory to safely process more logs.*

*Get the spec of DaemonSet running in your cluster:*

```
kubectl get ds fluentd-gcp-v2.0 --namespace kube-system -o
yaml > fluentd-gcp-ds.yaml
```

*Then edit resource requirements in the spec file and update the DaemonSet object in the apiserver using the following command:*

```
kubectl replace -f fluentd-gcp-ds.yaml
```

*After some time, Stackdriver Logging agent pods will be restarted with the new configuration.*

## **Changing fluentd parameters**

Fluentd configuration is stored in the *ConfigMap* object. It is effectively a set of configuration files that are merged together. You can learn about fluentd configuration on the [official site](#).

Imagine you want to add a new parsing logic to the configuration, so that fluentd can understand default Python logging format. An appropriate fluentd filter looks similar to this:

```
<filter reform.**>
  type parser
  format /^(<severity>\w) : (<logger_name>\w) : (<log>.*/)
  reserve_data true
  suppress_parse_error_log true
  key_name log
</filter>
```

Now you have to put it in the configuration and make Stackdriver Logging agents pick it up. Get the current version of the Stackdriver Logging *ConfigMap* in your cluster by running the following command:

```
kubectl get cm fluentd-gcp-config --namespace kube-system -o yaml > fluentd-gcp-configmap.yaml
```

Then in the value of the key *containers.input.conf* insert a new filter right after the *source* section.

**Note:** Order is important.

Updating *ConfigMap* in the *apiserver* is more complicated than updating *DaemonSet*. It's better to consider *ConfigMap* to be immutable. Then, in order to update the configuration, you should create *ConfigMap* with a new name and then change *DaemonSet* to point to it using [guide above](#).

## **Adding fluentd plugins**

Fluentd is written in Ruby and allows to extend its capabilities using [plugins](#). If you want to use a plugin, which is not included in the default Stackdriver Logging container image, you have to build a custom image. Imagine you want to add Kafka sink for messages from a particular container for additional processing. You can re-use the default [container image sources](#) with minor changes:

- Change Makefile to point to your container repository, for example `PREFIX=gcr.io/<your-project-id>`.
- Add your dependency to the Gemfile, for example `gem 'fluent-plugin-kafka'`.

Then run `make build push` from this directory. After updating *DaemonSet* to pick up the new image, you can use the plugin you installed in the fluentd configuration.

# 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](#).

Last modified August 07, 2020 at 4:46 PM PST: [Tune links in tasks section \(1/2\) \(b8541d212\)](#)

[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Deploying](#)
  - [Deploying to a new cluster](#)
  - [Deploying to an existing cluster](#)
- [Verifying your Logging Agent Deployment](#)
- [Viewing logs](#)
  - [Exporting logs](#)
- [Configuring Stackdriver Logging Agents](#)
  - [Prerequisites](#)
  - [Changing DaemonSet parameters](#)
  - [Changing fluentd parameters](#)
  - [Adding fluentd plugins](#)

# Monitor Node Health

Node problem detector is a [DaemonSet](#) monitoring the node health. It collects node problems from various daemons and reports them to the apiserver as [NodeCondition](#) and [Event](#).

It supports some known kernel issue detection now, and will detect more and more node problems over time.

Currently Kubernetes won't take any action on the node conditions and events generated by node problem detector. In the future, a remedy system could be introduced to deal with node problems.

See more information [here](#).

# Before you begin

You need to have a Kubernetes cluster, and the kubectl command-line tool must be configured to communicate with your cluster. If you do not already have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:

- [Katacoda](#)
- [Play with Kubernetes](#)

To check the version, enter `kubectl version`.

## **Limitations**

- The kernel issue detection of node problem detector only supports file based kernel log now. It doesn't support log tools like journald.
- The kernel issue detection of node problem detector has assumption on kernel log format, and now it only works on Ubuntu and Debian. However, it is easy to extend it to [support other log format](#).

## **Enable/Disable in GCE cluster**

Node problem detector is [running as a cluster addon](#) enabled by default in the gce cluster.

You can enable/disable it by setting the environment variable `KUBE_ENABLE_NODE_PROBLEM_DETECTOR` before `kube-up.sh`.

## **Use in Other Environment**

To enable node problem detector in other environment outside of GCE, you can use either `kubectl` or `addon pod`.

### **Kubectl**

This is the recommended way to start node problem detector outside of GCE. It provides more flexible management, such as overwriting the default configuration to fit it into your environment or detect customized node problems.

- **Step 1:** `node-problem-detector.yaml`:  
[debug/node-problem-detector.yaml](#)  


```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: node-problem-detector-v0.1
  namespace: kube-system
  labels:
    k8s-app: node-problem-detector
```

```

version: v0.1
kubernetes.io/cluster-service: "true"
spec:
  selector:
    matchLabels:
      k8s-app: node-problem-detector
      version: v0.1
      kubernetes.io/cluster-service: "true"
  template:
    metadata:
      labels:
        k8s-app: node-problem-detector
        version: v0.1
        kubernetes.io/cluster-service: "true"
    spec:
      hostNetwork: true
      containers:
        - name: node-problem-detector
          image: k8s.gcr.io/node-problem-detector:v0.1
          securityContext:
            privileged: true
          resources:
            limits:
              cpu: "200m"
              memory: "100Mi"
            requests:
              cpu: "20m"
              memory: "20Mi"
          volumeMounts:
            - name: log
              mountPath: /log
              readOnly: true
          volumes:
            - name: log
              hostPath:
                path: /var/log/

```

**Notice that you should make sure the system log directory is right for your OS distro.**

- **Step 2:** Start node problem detector with kubectl:

```
kubectl apply -f https://k8s.io/examples/debug/node-problem-detector.yaml
```

## Addon Pod

*This is for those who have their own cluster bootstrap solution, and don't need to overwrite the default configuration. They could leverage the addon pod to further automate the deployment.*

*Just create `node-problem-detector.yaml`, and put it under the addon pods directory `/etc/kubernetes/addons/node-problem-detector` on master node.*

## Overwrite the Configuration

*The [default configuration](#) is embedded when building the Docker image of node problem detector.*

However, you can use [ConfigMap](#) to overwrite it following the steps:

- **Step 1:** Change the config files in `config/`.
- **Step 2:** Create the ConfigMap `node-problem-detector-config` with `kubectl create configmap node-problem-detector-config --from-file=config/`.
- **Step 3:** Change the `node-problem-detector.yaml` to use the ConfigMap:

[debug/node-problem-detector-configmap.yaml](#)  


```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: node-problem-detector-v0.1
  namespace: kube-system
  labels:
    k8s-app: node-problem-detector
    version: v0.1
    kubernetes.io/cluster-service: "true"
spec:
  selector:
    matchLabels:
      k8s-app: node-problem-detector
      version: v0.1
      kubernetes.io/cluster-service: "true"
  template:
    metadata:
      labels:
        k8s-app: node-problem-detector
        version: v0.1
        kubernetes.io/cluster-service: "true"
    spec:
      hostNetwork: true
      containers:
```

```

- name: node-problem-detector
  image: k8s.gcr.io/node-problem-detector:v0.1
  securityContext:
    privileged: true
  resources:
    limits:
      cpu: "200m"
      memory: "100Mi"
    requests:
      cpu: "20m"
      memory: "20Mi"
  volumeMounts:
    - name: log
      mountPath: /log
      readOnly: true
    - name: config # Overwrite the config/ directory
      with ConfigMap volume
        mountPath: /config
        readOnly: true
  volumes:
    - name: log
      hostPath:
        path: /var/log/
    - name: config # Define ConfigMap volume
      configMap:
        name: node-problem-detector-config

```

- **Step 4:** Re-create the node problem detector with the new yaml file:

```

kubectl delete -f https://k8s.io/examples/debug/node-
problem-detector.yaml # If you have a node-problem-detector
running
kubectl apply -f https://k8s.io/examples/debug/node-problem-
detector-configmap.yaml

```

**Notice that this approach only applies to node problem detector started with kubectl.**

*For node problem detector running as cluster addon, because addon manager doesn't support ConfigMap, configuration overwriting is not supported now.*

## Kernel Monitor

*Kernel Monitor is a problem daemon in node problem detector. It monitors kernel log and detects known kernel issues following predefined rules.*

The Kernel Monitor matches kernel issues according to a set of predefined rule list in [config/kernel-monitor.json](#). The rule list is extensible, and you can always extend it by overwriting the configuration.

## Add New NodeConditions

To support new node conditions, you can extend the `conditions` field in `config/kernel-monitor.json` with new condition definition:

```
{  
  "type": "NodeConditionType",  
  "reason": "CamelCaseDefaultNodeConditionReason",  
  "message": "arbitrary default node condition message"  
}
```

## Detect New Problems

To detect new problems, you can extend the `rules` field in `config/kernel-monitor.json` with new rule definition:

```
{  
  "type": "temporary/permanent",  
  "condition": "NodeConditionOfPermanentIssue",  
  "reason": "CamelCaseShortReason",  
  "message": "regexp matching the issue in the kernel log"  
}
```

## Change Log Path

Kernel log in different OS distros may locate in different path. The `log` field in `config/kernel-monitor.json` is the log path inside the container. You can always configure it to match your OS distro.

## Support Other Log Format

Kernel monitor uses [Translator](#) plugin to translate kernel log the internal data structure. It is easy to implement a new translator for a new log format.

## Caveats

*It is recommended to run the node problem detector in your cluster to monitor the node health. However, you should be aware that this will introduce extra resource overhead on each node. Usually this is fine, because:*

- *The kernel log is generated relatively slowly.*
- *Resource limit is set for node problem detector.*
- *Even under high load, the resource usage is acceptable. (see [benchmark result](#))*

## 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](#).

Last modified May 30, 2020 at 3:10 PM PST: [add en pages \(ecc27bbbe\)](#)  
[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Limitations](#)
- [Enable/Disable in GCE cluster](#)
- [Use in Other Environment](#)
  - [Kubectl](#)
  - [Addon Pod](#)
- [Overwrite the Configuration](#)
- [Kernel Monitor](#)
  - [Add New NodeConditions](#)
  - [Detect New Problems](#)
  - [Change Log Path](#)
  - [Support Other Log Format](#)
- [Caveats](#)

## Resource metrics pipeline

Resource usage metrics, such as container CPU and memory usage, are available in Kubernetes through the Metrics API. These metrics can be accessed either directly by the user with the `kubectl top` command, or by a controller in the cluster, for example Horizontal Pod Autoscaler, to make decisions.

## The Metrics API

Through the Metrics API, you can get the amount of resource currently used by a given node or a given pod. This API doesn't store the metric values, so it's not possible, for example, to get the amount of resources used by a given node 10 minutes ago.

The API is no different from any other API:

- it is discoverable through the same endpoint as the other Kubernetes APIs under the path: `/apis/metrics.k8s.io/`
- it offers the same security, scalability, and reliability guarantees

The API is defined in [k8s.io/metrics](#) repository. You can find more information about the API there.

**Note:** The API requires the metrics server to be deployed in the cluster. Otherwise it will be not available.

## Measuring Resource Usage

### CPU

CPU is reported as the average usage, in [CPU cores](#), over a period of time. This value is derived by taking a rate over a cumulative CPU counter provided by the kernel (in both Linux and Windows kernels). The kubelet chooses the window for the rate calculation.

### Memory

Memory is reported as the working set, in bytes, at the instant the metric was collected. In an ideal world, the "working set" is the amount of memory in-use that cannot be freed under memory pressure. However, calculation of the working set varies by host OS, and generally makes heavy use of heuristics to produce an estimate. It includes all anonymous (non-file-backed) memory since Kubernetes does not support swap. The metric typically also includes some cached (file-backed) memory, because the host OS cannot always reclaim such pages.

## Metrics Server

[Metrics Server](#) is a cluster-wide aggregator of resource usage data. By default, it is deployed in clusters created by `kube-up.sh` script as a Deployment object. If you use a different Kubernetes setup mechanism, you can deploy it using the provided [deployment components.yaml](#) file.

Metrics Server collects metrics from the Summary API, exposed by [Kubelet](#) on each node, and is registered with the main API server via [Kubernetes aggregator](#).

Learn more about the metrics server in [the design doc](#).

## 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](#).

Last modified October 17, 2020 at 3:21 PM PST: [update kubernetes-incubator references \(a8b6551c2\)](#)

[Edit this page](#) [Create child page](#) [Create an issue](#)

- [The Metrics API](#)
- [Measuring Resource Usage](#)
  - [CPU](#)
  - [Memory](#)
- [Metrics Server](#)

## Tools for Monitoring Resources

To scale an application and provide a reliable service, you need to understand how the application behaves when it is deployed. You can examine application performance in a Kubernetes cluster by examining the containers, [pods](#), [services](#), and the characteristics of the overall cluster. Kubernetes provides detailed information about an application's resource usage at each of these levels. This information allows you to evaluate your application's performance and where bottlenecks can be removed to improve overall performance.

In Kubernetes, application monitoring does not depend on a single monitoring solution. On new clusters, you can use [resource metrics](#) or [full metrics](#) pipelines to collect monitoring statistics.

### Resource metrics pipeline

The resource metrics pipeline provides a limited set of metrics related to cluster components such as the [Horizontal Pod Autoscaler](#) controller, as well as the `kubectl top` utility. These metrics are collected by the lightweight, short-term, in-memory [metrics-server](#) and are exposed via the `metrics.k8s.io` API.

metrics-server discovers all nodes on the cluster and queries each node's [kubelet](#) for CPU and memory usage. The kubelet acts as a bridge between the Kubernetes master and the nodes, managing the pods and containers running on a machine. The kubelet translates each pod into its constituent containers and fetches individual container usage

*statistics from the container runtime through the container runtime interface. The kubelet fetches this information from the integrated cAdvisor for the legacy Docker integration. It then exposes the aggregated pod resource usage statistics through the metrics-server Resource Metrics API. This API is served at /metrics/resource/v1beta1 on the kubelet's authenticated and read-only ports.*

## **Full metrics pipeline**

*A full metrics pipeline gives you access to richer metrics. Kubernetes can respond to these metrics by automatically scaling or adapting the cluster based on its current state, using mechanisms such as the Horizontal Pod Autoscaler. The monitoring pipeline fetches metrics from the kubelet and then exposes them to Kubernetes via an adapter by implementing either the custom.metrics.k8s.io or external.metrics.k8s.io API.*

*Prometheus, a CNCF project, can natively monitor Kubernetes, nodes, and Prometheus itself. Full metrics pipeline projects that are not part of the CNCF are outside the scope of Kubernetes documentation.*

## **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](#).*

*Last modified October 17, 2020 at 3:21 PM PST: [update kubernetes-incubator references \(a8b6551c2\)](#)*

*[Edit this page](#) [Create child page](#) [Create an issue](#)*

- [Resource metrics pipeline](#)*
- [Full metrics pipeline](#)*

## **Troubleshoot Applications**

*This guide is to help users debug applications that are deployed into Kubernetes and not behaving correctly. This is not a guide for people who want to debug their cluster. For that you should check out [this guide](#).*

# **Diagnosing the problem**

*The first step in troubleshooting is triage. What is the problem? Is it your Pods, your Replication Controller or your Service?*

- [Debugging Pods](#)
- [Debugging Replication Controllers](#)
- [Debugging Services](#)

## **Debugging Pods**

*The first step in debugging a Pod is taking a look at it. Check the current state of the Pod and recent events with the following command:*

```
kubectl describe pods ${POD_NAME}
```

*Look at the state of the containers in the pod. Are they all Running? Have there been recent restarts?*

*Continue debugging depending on the state of the pods.*

### **My pod stays pending**

*If a Pod is stuck in Pending it means that it can not be scheduled onto a node. Generally this is because there are insufficient resources of one type or another that prevent scheduling. Look at the output of the `kube ctl describe ...` command above. There should be messages from the scheduler about why it can not schedule your pod. Reasons include:*

- **You don't have enough resources:** You may have exhausted the supply of CPU or Memory in your cluster, in this case you need to delete Pods, adjust resource requests, or add new nodes to your cluster. See [Compute Resources document](#) for more information.
- **You are using hostPort:** When you bind a Pod to a hostPort there are a limited number of places that pod can be scheduled. In most cases, hostPort is unnecessary, try using a Service object to expose your Pod. If you do require hostPort then you can only schedule as many Pods as there are nodes in your Kubernetes cluster.

### **My pod stays waiting**

*If a Pod is stuck in the Waiting state, then it has been scheduled to a worker node, but it can't run on that machine. Again, the information from `kubectl describe ...` should be informative. The most common cause of Waiting pods is a failure to pull the image. There are three things to check:*

- Make sure that you have the name of the image correct.
- Have you pushed the image to the repository?

- Run a manual `docker pull <image>` on your machine to see if the image can be pulled.

## **My pod is crashing or otherwise unhealthy**

Once your pod has been scheduled, the methods described in [Debug Running Pods](#) are available for debugging.

## **My pod is running but not doing what I told it to do**

If your pod is not behaving as you expected, it may be that there was an error in your pod description (e.g. `mypod.yaml` file on your local machine), and that the error was silently ignored when you created the pod. Often a section of the pod description is nested incorrectly, or a key name is typed incorrectly, and so the key is ignored. For example, if you misspelled `command` as `commnd` then the pod will be created but will not use the command line you intended it to use.

The first thing to do is to delete your pod and try creating it again with the `--validate` option. For example, run `kubectl apply --validate -f mypod.yaml`. If you misspelled `command` as `commnd` then will give an error like this:

```
I0805 10:43:25.129850    46757 schema.go:126] unknown field:  
commnd  
I0805 10:43:25.129973    46757 schema.go:129] this may be a fa  
lse alarm, see https://github.com/kubernetes/kubernetes/  
issues/6842  
pods/mypod
```

The next thing to check is whether the pod on the apiserver matches the pod you meant to create (e.g. in a yaml file on your local machine). For example, run `kubectl get pods/mypod -o yaml > mypod-on-apiserver.yaml` and then manually compare the original pod description, `mypod.yaml` with the one you got back from apiserver, `mypod-on-apiserver.yaml`. There will typically be some lines on the "apiserver" version that are not on the original version. This is expected. However, if there are lines on the original that are not on the apiserver version, then this may indicate a problem with your pod spec.

## **Debugging Replication Controllers**

Replication controllers are fairly straightforward. They can either create Pods or they can't. If they can't create pods, then please refer to the [instructions above](#) to debug your pods.

You can also use `kubectl describe rc ${CONTROLLER_NAME}` to introspect events related to the replication controller.

## **Debugging Services**

Services provide load balancing across a set of pods. There are several common problems that can make Services not work properly. The following instructions should help debug Service problems.

First, verify that there are endpoints for the service. For every Service object, the apiserver makes an `endpoints` resource available.

You can view this resource with:

```
kubectl get endpoints ${SERVICE_NAME}
```

Make sure that the endpoints match up with the number of pods that you expect to be members of your service. For example, if your Service is for an nginx container with 3 replicas, you would expect to see three different IP addresses in the Service's endpoints.

### **My service is missing endpoints**

If you are missing endpoints, try listing pods using the labels that Service uses. Imagine that you have a Service where the labels are:

```
...
spec:
  selector:
    name: nginx
    type: frontend
```

You can use:

```
kubectl get pods --selector=name=nginx,type=frontend
```

to list pods that match this selector. Verify that the list matches the Pods that you expect to provide your Service.

If the list of pods matches expectations, but your endpoints are still empty, it's possible that you don't have the right ports exposed. If your service has a `containerPort` specified, but the Pods that are selected don't have that port listed, then they won't be added to the endpoints list.

Verify that the pod's `containerPort` matches up with the Service's `targetPort`

### **Network traffic is not forwarded**

If you can connect to the service, but the connection is immediately dropped, and there are endpoints in the endpoints list, it's likely that the proxy can't contact your pods.

*There are three things to check:*

- Are your pods working correctly? Look for restart count, and [debug pods](#).
- Can you connect to your pods directly? Get the IP address for the Pod, and try to connect directly to that IP.
- Is your application serving on the port that you configured? Kubernetes doesn't do port remapping, so if your application serves on 8080, the `containerPort` field needs to be 8080.

## What's next

*If none of the above solves your problem, follow the instructions in [Debugging Service document](#) to make sure that your Service is running, has Endpoints, and your Pods are actually serving; you have DNS working, iptables rules installed, and kube-proxy does not seem to be misbehaving.*

*You may also visit [troubleshooting document](#) 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](#).*

*Last modified August 07, 2020 at 4:46 PM PST: [Tune links in tasks section \(1/2\) \(b8541d212\)](#)*

[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Diagnosing the problem](#)
  - [Debugging Pods](#)
  - [Debugging Replication Controllers](#)
  - [Debugging Services](#)
- [What's next](#)

## Troubleshoot Clusters

*This doc is about cluster troubleshooting; we assume you have already ruled out your application as the root cause of the problem you are experiencing. See the [application troubleshooting guide](#) for tips on application debugging. You may also visit [troubleshooting document](#) for more information.*

## ***Listing your cluster***

*The first thing to debug in your cluster is if your nodes are all registered correctly.*

*Run*

```
kubectl get nodes
```

*And verify that all of the nodes you expect to see are present and that they are all in the Ready state.*

*To get detailed information about the overall health of your cluster, you can run:*

```
kubectl cluster-info dump
```

## ***Looking at logs***

*For now, digging deeper into the cluster requires logging into the relevant machines. Here are the locations of the relevant log files. (note that on systemd-based systems, you may need to use journalctl instead)*

### ***Master***

- */var/log/kube-apiserver.log - API Server, responsible for serving the API*
- */var/log/kube-scheduler.log - Scheduler, responsible for making scheduling decisions*
- */var/log/kube-controller-manager.log - Controller that manages replication controllers*

### ***Worker Nodes***

- */var/log/kubelet.log - Kubelet, responsible for running containers on the node*
- */var/log/kube-proxy.log - Kube Proxy, responsible for service load balancing*

## ***A general overview of cluster failure modes***

*This is an incomplete list of things that could go wrong, and how to adjust your cluster setup to mitigate the problems.*

### ***Root causes:***

- *VM(s) shutdown*
- *Network partition within cluster, or between cluster and users*
- *Crashes in Kubernetes software*

- Data loss or unavailability of persistent storage (e.g. GCE PD or AWS EBS volume)
- Operator error, for example misconfigured Kubernetes software or application software

## **Specific scenarios:**

- Apiserver VM shutdown or apiserver crashing
  - Results
    - unable to stop, update, or start new pods, services, replication controller
    - existing pods and services should continue to work normally, unless they depend on the Kubernetes API
- Apiserver backing storage lost
  - Results
    - apiserver should fail to come up
    - kubelets will not be able to reach it but will continue to run the same pods and provide the same service proxying
    - manual recovery or recreation of apiserver state necessary before apiserver is restarted
- Supporting services (node controller, replication controller manager, scheduler, etc) VM shutdown or crashes
  - currently those are colocated with the apiserver, and their unavailability has similar consequences as apiserver
  - in future, these will be replicated as well and may not be co-located
  - they do not have their own persistent state
- Individual node (VM or physical machine) shuts down
  - Results
    - pods on that Node stop running
- Network partition
  - Results
    - partition A thinks the nodes in partition B are down; partition B thinks the apiserver is down. (Assuming the master VM ends up in partition A.)
- Kubelet software fault
  - Results
    - crashing kubelet cannot start new pods on the node
    - kubelet might delete the pods or not
    - node marked unhealthy
    - replication controllers start new pods elsewhere
- Cluster operator error
  - Results
    - loss of pods, services, etc
    - lost of apiserver backing store
    - users unable to read API
    - etc.

## **Mitigations:**

- Action: Use IaaS provider's automatic VM restarting feature for IaaS VMs
  - Mitigates: Apiserver VM shutdown or apiserver crashing
  - Mitigates: Supporting services VM shutdown or crashes
- Action: Use IaaS providers reliable storage (e.g. GCE PD or AWS EBS volume) for VMs with apiserver+etcd
  - Mitigates: Apiserver backing storage lost
- Action: Use [high-availability](#) configuration
  - Mitigates: Control plane node shutdown or control plane components (scheduler, API server, controller-manager) crashing
    - Will tolerate one or more simultaneous node or component failures
  - Mitigates: API server backing storage (i.e., etcd's data directory) lost
    - Assumes HA (highly-available) etcd configuration
- Action: Snapshot apiserver PDs/EBS-volumes periodically
  - Mitigates: Apiserver backing storage lost
  - Mitigates: Some cases of operator error
  - Mitigates: Some cases of Kubernetes software fault
- Action: use replication controller and services in front of pods
  - Mitigates: Node shutdown
  - Mitigates: Kubelet software fault
- Action: applications (containers) designed to tolerate unexpected restarts
  - Mitigates: Node shutdown
  - Mitigates: Kubelet software fault

## **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](#).

Last modified August 07, 2020 at 4:46 PM PST: [Tune links in tasks section \(1/2\) \(b8541d212\)](#)

[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Listing your cluster](#)
- [Looking at logs](#)
  - [Master](#)
  - [Worker Nodes](#)
- [A general overview of cluster failure modes](#)
  - [Root causes:](#)
  - [Specific scenarios:](#)
  - [Mitigations:](#)

# Troubleshooting

Sometimes things go wrong. This guide is aimed at making them right. It has two sections:

- [Troubleshooting your application](#) - Useful for users who are deploying code into Kubernetes and wondering why it is not working.
- [Troubleshooting your cluster](#) - Useful for cluster administrators and people whose Kubernetes cluster is unhappy.

You should also check the known issues for the [release](#) you're using.

## Getting help

If your problem isn't answered by any of the guides above, there are variety of ways for you to get help from the Kubernetes community.

### Questions

The documentation on this site has been structured to provide answers to a wide range of questions. [Concepts](#) explain the Kubernetes architecture and how each component works, while [Setup](#) provides practical instructions for getting started. [Tasks](#) show how to accomplish commonly used tasks, and [Tutorials](#) are more comprehensive walkthroughs of real-world, industry-specific, or end-to-end development scenarios. The [Reference](#) section provides detailed documentation on the [Kubernetes API](#) and command-line interfaces (CLIs), such as [kubectl](#).

# **Help! My question isn't covered! I need help now!**

## **Stack Overflow**

*Someone else from the community may have already asked a similar question or may be able to help with your problem. The Kubernetes team will also monitor [posts tagged Kubernetes](#). If there aren't any existing questions that help, please [ask a new one!](#)*

## **Slack**

*Many people from the Kubernetes community hang out on Kubernetes Slack in the #kubernetes-users channel. Slack requires registration; you can [request an invitation](#), and registration is open to everyone). Feel free to come and ask any and all questions. Once registered, access the [Kubernetes organisation in Slack](#) via your web browser or via Slack's own dedicated app.*

*Once you are registered, browse the growing list of channels for various subjects of interest. For example, people new to Kubernetes may also want to join the [#kubernetes-novice](#) channel. As another example, developers should join the [#kubernetes-dev](#) channel.*

*There are also many country specific / local language channels. Feel free to join these channels for localized support and info:*

<b>Country</b>	<b>Channels</b>
China	<a href="#">#cn-users</a> , <a href="#">#cn-events</a>
Finland	<a href="#">#fi-users</a>
France	<a href="#">#fr-users</a> , <a href="#">#fr-events</a>
Germany	<a href="#">#de-users</a> , <a href="#">#de-events</a>
India	<a href="#">#in-users</a> , <a href="#">#in-events</a>
Italy	<a href="#">#it-users</a> , <a href="#">#it-events</a>
Japan	<a href="#">#jp-users</a> , <a href="#">#jp-events</a>
Korea	<a href="#">#kr-users</a>
Netherlands	<a href="#">#nl-users</a>
Norway	<a href="#">#norw-users</a>
Poland	<a href="#">#pl-users</a>
Russia	<a href="#">#ru-users</a>
Spain	<a href="#">#es-users</a>
Sweden	<a href="#">#se-users</a>
Turkey	<a href="#">#tr-users</a> , <a href="#">#tr-events</a>

## **Forum**

*You're welcome to join the official Kubernetes Forum:  
[discuss.kubernetes.io](#).*

## **Bugs and feature requests**

If you have what looks like a bug, or you would like to make a feature request, please use the [GitHub issue tracking system](#).

Before you file an issue, please search existing issues to see if your issue is already covered.

If filing a bug, please include detailed information about how to reproduce the problem, such as:

- Kubernetes version: `kubectl version`
- Cloud provider, OS distro, network configuration, and Docker version
- Steps to reproduce the problem

## **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](#).

Last modified August 12, 2020 at 9:30 AM PST: [Tidy troubleshooting task \(84b05958b\)](#)

[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Getting help](#)
  - [Questions](#)
  - [Help! My question isn't covered! I need help now!](#)
    - [Stack Overflow](#)
    - [Slack](#)
    - [Forum](#)
    - [Bugs and feature requests](#)

## **Extend Kubernetes**

Understand advanced ways to adapt your Kubernetes cluster to the needs of your work environment.

---

[Configure the Aggregation Layer](#)

[Use Custom Resources](#)

[Set up an Extension API Server](#)

[Configure Multiple Schedulers](#)

## [Use an HTTP Proxy to Access the Kubernetes API](#)

### [Set up Konnectivity service](#)

# Configure the Aggregation Layer

Configuring the [aggregation layer](#) allows the Kubernetes apiserver to be extended with additional APIs, which are not part of the core Kubernetes APIs.

## Before you begin

You need to have a Kubernetes cluster, and the `kubectl` command-line tool must be configured to communicate with your cluster. If you do not already have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:

- [Katacoda](#)
- [Play with Kubernetes](#)

To check the version, enter `kubectl version`.

**Note:** There are a few setup requirements for getting the aggregation layer working in your environment to support mutual TLS auth between the proxy and extension apiservers. Kubernetes and the kube-apiserver have multiple CAs, so make sure that the proxy is signed by the aggregation layer CA and not by something else, like the master CA.

**Caution:** Reusing the same CA for different client types can negatively impact the cluster's ability to function. For more information, see [CA Reusage and Conflicts](#).

## Authentication Flow

Unlike Custom Resource Definitions (CRDs), the Aggregation API involves another server - your Extension apiserver - in addition to the standard Kubernetes apiserver. The Kubernetes apiserver will need to communicate with your extension apiserver, and your extension apiserver will need to communicate with the Kubernetes apiserver. In order for this communication to be secured, the Kubernetes apiserver uses x509 certificates to authenticate itself to the extension apiserver.

This section describes how the authentication and authorization flows work, and how to configure them.

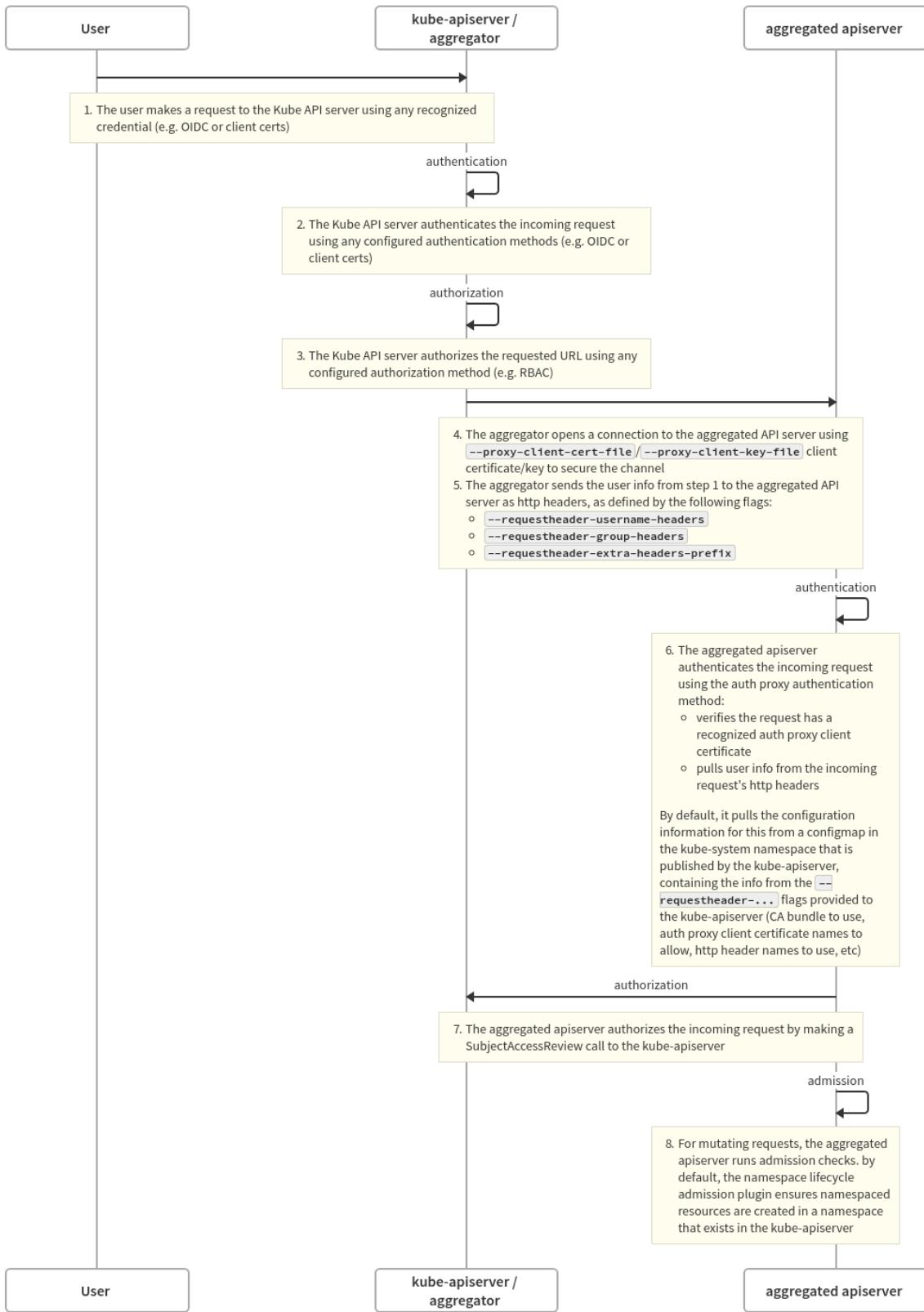
*The high-level flow is as follows:*

1. *Kubernetes apiserver: authenticate the requesting user and authorize their rights to the requested API path.*
2. *Kubernetes apiserver: proxy the request to the extension apiserver*
3. *Extension apiserver: authenticate the request from the Kubernetes apiserver*
4. *Extension apiserver: authorize the request from the original user*
5. *Extension apiserver: execute*

*The rest of this section describes these steps in detail.*

*The flow can be seen in the following diagram.*

## Welcome to swimlanes.io



POWERED BY [swimlanes.io](#)

*The source for the above swimlanes can be found in the source of this document.*

## **Kubernetes Apiserver Authentication and Authorization**

*A request to an API path that is served by an extension apiserver begins the same way as all API requests: communication to the Kubernetes apiserver. This path already has been registered with the Kubernetes apiserver by the extension apiserver.*

*The user communicates with the Kubernetes apiserver, requesting access to the path. The Kubernetes apiserver uses standard authentication and authorization configured with the Kubernetes apiserver to authenticate the user and authorize access to the specific path.*

*For an overview of authenticating to a Kubernetes cluster, see ["Authenticating to a Cluster"](#). For an overview of authorization of access to Kubernetes cluster resources, see ["Authorization Overview"](#).*

*Everything to this point has been standard Kubernetes API requests, authentication and authorization.*

*The Kubernetes apiserver now is prepared to send the request to the extension apiserver.*

## **Kubernetes Apiserver Proxies the Request**

*The Kubernetes apiserver now will send, or proxy, the request to the extension apiserver that registered to handle the request. In order to do so, it needs to know several things:*

1. *How should the Kubernetes apiserver authenticate to the extension apiserver, informing the extension apiserver that the request, which comes over the network, is coming from a valid Kubernetes apiserver?*
2. *How should the Kubernetes apiserver inform the extension apiserver of the username and group for which the original request was authenticated?*

*In order to provide for these two, you must configure the Kubernetes apiserver using several flags.*

## **Kubernetes Apiserver Client Authentication**

*The Kubernetes apiserver connects to the extension apiserver over TLS, authenticating itself using a client certificate. You must provide the following to the Kubernetes apiserver upon startup, using the provided flags:*

- private key file via `--proxy-client-key-file`
- signed client certificate file via `--proxy-client-cert-file`
- certificate of the CA that signed the client certificate file via `--requestheader-client-ca-file`
- valid Common Name values (CNs) in the signed client certificate via `--requestheader-allowed-names`

*The Kubernetes apiserver will use the files indicated by `--proxy-client-*-file` to authenticate to the extension apiserver. In order for the request to be considered valid by a compliant extension apiserver, the following conditions must be met:*

1. *The connection must be made using a client certificate that is signed by the CA whose certificate is in `--requestheader-client-ca-file`.*
2. *The connection must be made using a client certificate whose CN is one of those listed in `--requestheader-allowed-names`.*

**Note:** You can set this option to blank as `--requestheader-allowed-names=""`. This will indicate to an extension apiserver that any CN is acceptable.

*When started with these options, the Kubernetes apiserver will:*

1. *Use them to authenticate to the extension apiserver.*
2. *Create a configmap in the `kube-system` namespace called `extension-apiserver-authentication`, in which it will place the CA certificate and the allowed CNs. These in turn can be retrieved by extension apiservers to validate requests.*

*Note that the same client certificate is used by the Kubernetes apiserver to authenticate against all extension apiservers. It does not create a client certificate per extension apiserver, but rather a single one to authenticate as the Kubernetes apiserver. This same one is reused for all extension apiserver requests.*

## **Original Request Username and Group**

*When the Kubernetes apiserver proxies the request to the extension apiserver, it informs the extension apiserver of the username and group with which the original request successfully authenticated. It provides these in http headers of its proxied request. You must inform the Kubernetes apiserver of the names of the headers to be used.*

- the header in which to store the username via `--requestheader-username-headers`
- the header in which to store the group via `--requestheader-group-headers`
- the prefix to append to all extra headers via `--requestheader-extra-headers-prefix`

*These header names are also placed in the `extension-apiserver-authentication configmap`, so they can be retrieved and used by extension apiservers.*

## ***Extension Apiserver Authenticates the Request***

*The extension apiserver, upon receiving a proxied request from the Kubernetes apiserver, must validate that the request actually did come from a valid authenticating proxy, which role the Kubernetes apiserver is fulfilling. The extension apiserver validates it via:*

1. *Retrieve the following from the configmap in `kube-system`, as described above:*
  - Client CA certificate
  - List of allowed names (CNs)
  - Header names for username, group and extra info
2. *Check that the TLS connection was authenticated using a client certificate which:*
  - Was signed by the CA whose certificate matches the retrieved CA certificate.
  - Has a CN in the list of allowed CNs, unless the list is blank, in which case all CNs are allowed.
  - Extract the username and group from the appropriate headers

*If the above passes, then the request is a valid proxied request from a legitimate authenticating proxy, in this case the Kubernetes apiserver.*

*Note that it is the responsibility of the extension apiserver implementation to provide the above. Many do it by default, leveraging the `k8s.io/apiserver/` package. Others may provide options to override it using command-line options.*

*In order to have permission to retrieve the configmap, an extension apiserver requires the appropriate role. There is a default role named `extension-apiserver-authentication-reader` in the `kube-system` namespace which can be assigned.*

## ***Extension Apiserver Authorizes the Request***

The extension apiserver now can validate that the user/group retrieved from the headers are authorized to execute the given request. It does so by sending a standard [SubjectAccessReview](#) request to the Kubernetes apiserver.

In order for the extension apiserver to be authorized itself to submit the [SubjectAccessReview](#) request to the Kubernetes apiserver, it needs the correct permissions. Kubernetes includes a default ClusterRole named `system:auth-delegator` that has the appropriate permissions. It can be granted to the extension apiserver's service account.

## **Extension Apiserver Executes**

If the [SubjectAccessReview](#) passes, the extension apiserver executes the request.

## **Enable Kubernetes Apiserver flags**

Enable the aggregation layer via the following `kube-apiserver` flags. They may have already been taken care of by your provider.

```
--requestheader-client-ca-file=<path to aggregator CA cert>
--requestheader-allowed-names=front-proxy-client
--requestheader-extra-headers-prefix=X-Remote-Extra-
--requestheader-group-headers=X-Remote-Group
--requestheader-username-headers=X-Remote-User
--proxy-client-cert-file=<path to aggregator proxy cert>
--proxy-client-key-file=<path to aggregator proxy key>
```

## **CA Reusage and Conflicts**

The Kubernetes apiserver has two client CA options:

- `--client-ca-file`
- `--requestheader-client-ca-file`

Each of these functions independently and can conflict with each other, if not used correctly.

- `--client-ca-file`: When a request arrives to the Kubernetes apiserver, if this option is enabled, the Kubernetes apiserver checks the certificate of the request. If it is signed by one of the CA certificates in the file referenced by `--client-ca-file`, then the request is treated as a legitimate request, and the user is the

- value of the common name `CN=`, while the group is the organization `O=`. See the [documentation on TLS authentication](#).*
- `--requestheader-client-ca-file`: When a request arrives to the Kubernetes apiserver, if this option is enabled, the Kubernetes apiserver checks the certificate of the request. If it is signed by one of the CA certificates in the file reference by `--requestheader-client-ca-file`, then the request is treated as a potentially legitimate request. The Kubernetes apiserver then checks if the common name `CN=` is one of the names in the list provided by `--requestheader-allowed-names`. If the name is allowed, the request is approved; if it is not, the request is not.

*If both `--client-ca-file` and `--requestheader-client-ca-file` are provided, then the request first checks the `--requestheader-client-ca-file` CA and then the `--client-ca-file`. Normally, different CAs, either root CAs or intermediate CAs, are used for each of these options; regular client requests match against `--client-ca-file`, while aggregation requests match against `--requestheader-client-ca-file`. However, if both use the same CA, then client requests that normally would pass via `--client-ca-file` will fail, because the CA will match the CA in `--requestheader-client-ca-file`, but the common name `CN=` will **not** match one of the acceptable common names in `--requestheader-allowed-names`. This can cause your kubelets and other control plane components, as well as end-users, to be unable to authenticate to the Kubernetes apiserver.*

*For this reason, use different CA certs for the `--client-ca-file` option - to authorize control plane components and end-users - and the `--requestheader-client-ca-file` option - to authorize aggregation apiserver requests.*

**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.

*If you are not running kube-proxy on a host running the API server, then you must make sure that the system is enabled with the following `kube-apiserver` flag:*

```
--enable-aggregator-routing=true
```

## **Register APIService objects**

*You can dynamically configure what client requests are proxied to extension apiserver. The following is an example registration:*

```
apiVersion: apiregistration.k8s.io/v1
kind: APIService
metadata:
  name: <name of the registration object>
spec:
  group: <API group name this extension apiserver hosts>
  version: <API version this extension apiserver hosts>
  groupPriorityMinimum: <priority this APIService for this group, see API documentation>
  versionPriority: <prioritizes ordering of this version within a group, see API documentation>
  service:
    namespace: <namespace of the extension apiserver service>
    name: <name of the extension apiserver service>
    caBundle: <pem encoded ca cert that signs the server cert used by the webhook>
```

The name of an APIService object must be a valid [path segment name](#).

## Contacting the extension apiserver

Once the Kubernetes apiserver has determined a request should be sent to an extension apiserver, it needs to know how to contact it.

The `service` stanza is a reference to the service for an extension apiserver. The `service namespace` and `name` are required. The `port` is optional and defaults to 443.

Here is an example of an extension apiserver that is configured to be called on port "1234", and to verify the TLS connection against the ServerName `my-service-name.my-service-namespace.svc` using a custom CA bundle.

```
apiVersion: apiregistration.k8s.io/v1
kind: APIService
...
spec:
  ...
  service:
    namespace: my-service-namespace
    name: my-service-name
    port: 1234
    caBundle: "Ci0tLS0tQk...<base64-encoded PEM
bundle>...tLS0K"
  ...
```

## What's next

- [Setup an extension api-server](#) to work with the aggregation layer.
- For a high level overview, see [Extending the Kubernetes API with the aggregation layer](#).
- Learn how to [Extend the Kubernetes API Using Custom Resource Definitions](#).

## 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](#).

Last modified May 04, 2020 at 6:56 PM PST: [move setup konnectivity svc \(5fe8c3ca5\)](#)

[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Authentication Flow](#)
  - [Kubernetes Apiserver Authentication and Authorization](#)
  - [Kubernetes Apiserver Proxies the Request](#)
  - [Extension Apiserver Authenticates the Request](#)
  - [Extension Apiserver Authorizes the Request](#)
  - [Extension Apiserver Executes](#)
- [Enable Kubernetes Apiserver flags](#)
  - [CA Reusage and Conflicts](#)
  - [Register APIService objects](#)
- [What's next](#)

## Use Custom Resources

---

[Extend the Kubernetes API with CustomResourceDefinitions](#)

[Versions in CustomResourceDefinitions](#)

# Extend the Kubernetes API with CustomResourceDefinitions

This page shows how to install a [custom resource](#) into the Kubernetes API by creating a [CustomResourceDefinition](#).

# **Before you begin**

You need to have a Kubernetes cluster, and the `kubectl` command-line tool must be configured to communicate with your cluster. If you do not already have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:

- [Katacoda](#)
- [Play with Kubernetes](#)

Your Kubernetes server must be at or later than version 1.16. To check the version, enter `kubectl version`. If you are using an older version of Kubernetes that is still supported, switch to the documentation for that version to see advice that is relevant for your cluster.

## **Create a CustomResourceDefinition**

When you create a new CustomResourceDefinition (CRD), the Kubernetes API Server creates a new RESTful resource path for each version you specify. The CRD can be either namespaced or cluster-scoped, as specified in the CRD's `scope` field. As with existing built-in objects, deleting a namespace deletes all custom objects in that namespace. CustomResourceDefinitions themselves are non-namespaced and are available to all namespaces.

For example, if you save the following CustomResourceDefinition to `resourcedefinition.yaml`:

```
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  # name must match the spec fields below, and be in the
  form: <plural>.<group>
  name: crontabs.stable.example.com
spec:
  # group name to use for REST API: /apis/<group>/<version>
  group: stable.example.com
  # list of versions supported by this
  # CustomResourceDefinition
  versions:
    - name: v1
      # Each version can be enabled/disabled by Served flag.
      served: true
      # One and only one version must be marked as the
      # storage version.
      storage: true
      schema:
        openAPIV3Schema:
          type: object
```

```

properties:
  spec:
    type: object
    properties:
      cronSpec:
        type: string
      image:
        type: string
      replicas:
        type: integer
    # either Namespaced or Cluster
  scope: Namespaced
  names:
    # plural name to be used in the URL: /apis/<group>/<version>/<plural>
    plural: crontabs
    # singular name to be used as an alias on the CLI and
    for display
    singular: crontab
    # kind is normally the CamelCased singular type. Your
    resource manifests use this.
    kind: CronTab
    # shortNames allow shorter string to match your resource
    on the CLI
    shortNames:
      - ct

```

and create it:

```
kubectl apply -f resourcedefinition.yaml
```

Then a new namespaced RESTful API endpoint is created at:

```
/apis/stable.example.com/v1/namespaces/*/crontabs/...
```

This endpoint URL can then be used to create and manage custom objects. The kind of these objects will be *CronTab* from the spec of the *CustomResourceDefinition* object you created above.

It might take a few seconds for the endpoint to be created. You can watch the *Established* condition of your *CustomResourceDefinition* to be true or watch the discovery information of the API server for your resource to show up.

## Create custom objects

After the `CustomResourceDefinition` object has been created, you can create custom objects. Custom objects can contain custom fields. These fields can contain arbitrary JSON. In the following example, the `cronSpec` and `image` custom fields are set in a custom object of kind `CronTab`. The kind `CronTab` comes from the spec of the `CustomResourceDefinition` object you created above.

If you save the following YAML to `my-crontab.yaml`:

```
apiVersion: "stable.example.com/v1"
kind: CronTab
metadata:
  name: my-new-cron-object
spec:
  cronSpec: "* * * * */5"
  image: my-awesome-cron-image
```

and create it:

```
kubectl apply -f my-crontab.yaml
```

You can then manage your `CronTab` objects using `kubectl`. For example:

```
kubectl get crontab
```

Should print a list like this:

NAME	AGE
my-new-cron-object	6s

Resource names are not case-sensitive when using `kubectl`, and you can use either the singular or plural forms defined in the CRD, as well as any short names.

You can also view the raw YAML data:

```
kubectl get ct -o yaml
```

You should see that it contains the custom `cronSpec` and `image` fields from the YAML you used to create it:

```
apiVersion: v1
kind: List
items:
- apiVersion: stable.example.com/v1
  kind: CronTab
  metadata:
    creationTimestamp: 2017-05-31T12:56:35Z
    generation: 1
    name: my-new-cron-object
    namespace: default
    resourceVersion: "285"
    uid: 9423255b-4600-11e7-af6a-28d2447dc82b
  spec:
    cronSpec: '* * * * */5'
    image: my-awesome-cron-image
metadata:
  resourceVersion: ""
```

## Delete a CustomResourceDefinition

When you delete a CustomResourceDefinition, the server will uninstall the RESTful API endpoint and delete all custom objects stored in it.

```
kubectl delete -f resourcedefinition.yaml
kubectl get crontabs
```

```
Error from server (NotFound): Unable to list
{"stable.example.com": "v1", "crontabs": "": the server could not
find the requested resource (get crontabs.stable.example.com)}
```

If you later recreate the same CustomResourceDefinition, it will start out empty.

## Specifying a structural schema

CustomResources store structured data in custom fields (alongside the built-in fields `apiVersion`, `kind` and `metadata`, which the API server validates implicitly). With [OpenAPI v3.0 validation](#) a schema can be specified, which is validated during creation and updates, compare below for details and limits of such a schema.

With `apiextensions.k8s.io/v1` the definition of a structural schema is mandatory for CustomResourceDefinitions. In the beta version of CustomResourceDefinition, the structural schema was optional.

A structural schema is an [OpenAPI v3.0 validation schema](#) which:

1. specifies a non-empty type (via `type` in OpenAPI) for the root, for each specified field of an object node (via `properties` or `additionalProperties` in OpenAPI) and for each item in an array node (via `items` in OpenAPI), with the exception of:
  - a node with `x-kubernetes-int-or-string: true`
  - a node with `x-kubernetes-preserve-unknown-fields: true`
2. for each field in an object and each item in an array which is specified within any of `allOf`, `anyOf`, `oneOf` or `not`, the schema also specifies the field/item outside of those logical junctors (compare example 1 and 2).
3. does not set `description`, `type`, `default`, `additionalProperties`, `nullable` within an `allOf`, `anyOf`, `oneOf` or `not`, with the exception of the two pattern for `x-kubernetes-int-or-string: true` (see below).
4. if `metadata` is specified, then only restrictions on `metadata.name` and `metadata.generateName` are allowed.

Non-structural example 1:

```
allOf:  
- properties:  
  foo:  
    ...  
  ...
```

conflicts with rule 2. The following would be correct:

```
properties:  
  foo:  
    ...  
allOf:  
- properties:  
  foo:  
    ...  
  ...
```

Non-structural example 2:

```
allOf:  
- items:  
  properties:  
    foo:  
    ...
```

conflicts with rule 2. The following would be correct:

```

items:
  properties:
    foo:
      ...
allOf:
- items:
  properties:
    foo:
      ...

```

*Non-structural example 3:*

```

properties:
  foo:
    pattern: "abc"
  metadata:
    type: object
    properties:
      name:
        type: string
        pattern: "^a"
    finalizers:
      type: array
      items:
        type: string
        pattern: "my-finalizer"
anyOf:
- properties:
  bar:
    type: integer
    minimum: 42
  required: ["bar"]
  description: "foo bar object"

```

*is not a structural schema because of the following violations:*

- the type at the root is missing (rule 1).
- the type of foo is missing (rule 1).
- bar inside of anyOf is not specified outside (rule 2).
- bar's type is within anyOf (rule 3).
- the description is set within anyOf (rule 3).
- metadata.finalizers might not be restricted (rule 4).

*In contrast, the following, corresponding schema is structural:*

```

type: object
description: "foo bar object"
properties:
  foo:

```

```
  type: string
  pattern: "abc"
  bar:
    type: integer
  metadata:
    type: object
  properties:
    name:
      type: string
      pattern: "^a"
anyOf:
- properties:
  bar:
    minimum: 42
required: ["bar"]
```

*Violations of the structural schema rules are reported in the `NonStructural` condition in the `CustomResourceDefinition`.*

## **Field pruning**

*CustomResourceDefinitions store validated resource data in the cluster's persistence store, [etcd](#). As with native Kubernetes resources such as [ConfigMap](#), if you specify a field that the API server does not recognize, the unknown field is pruned (removed) before being persisted.*

### **Note:**

*CRDs converted from `apiextensions.k8s.io/v1beta1` to `apiextensions.k8s.io/v1` might lack structural schemas, and `spec.preserveUnknownFields` might be `true`.*

*For migrated CustomResourceDefinitions where `spec.preserveUnknownFields` is set, pruning is not enabled and you can store arbitrary data. For best compatibility, you should update your custom resources to meet an OpenAPI schema, and you should set `spec.preserveUnknownFields` to `true` for the `CustomResourceDefinition` itself.*

*If you save the following YAML to `my-crontab.yaml`:*

```
apiVersion: "stable.example.com/v1"
kind: CronTab
metadata:
  name: my-new-cron-object
spec:
```

```
cronSpec: "* * * * */5"
image: my-awesome-cron-image
someRandomField: 42
```

and create it:

```
kubectl create --validate=false -f my-crontab.yaml -o yaml
```

your output is similar to:

```
apiVersion: stable.example.com/v1
kind: CronTab
metadata:
  creationTimestamp: 2017-05-31T12:56:35Z
  generation: 1
  name: my-new-cron-object
  namespace: default
  resourceVersion: "285"
  uid: 9423255b-4600-11e7-af6a-28d2447dc82b
spec:
  cronSpec: '* * * * */5'
  image: my-awesome-cron-image
```

Notice that the field `someRandomField` was pruned.

This example turned off client-side validation to demonstrate the API server's behavior, by adding the `--validate=false` command line option. Because the [OpenAPI validation schemas are also published](#) to clients, `kubectl` also checks for unknown fields and rejects those objects well before they would be sent to the API server.

## Controlling pruning

By default, all unspecified fields for a custom resource, across all versions, are pruned. It is possible though to opt-out of that for specific sub-trees of fields by adding `x-kubernetes-preserve-unknown-fields: true` in the [structural OpenAPI v3 validation schema](#). For example:

```
type: object
properties:
  json:
    x-kubernetes-preserve-unknown-fields: true
```

The field `json` can store any JSON value, without anything being pruned.

You can also partially specify the permitted JSON; for example:

```
type: object
properties:
  json:
    x-kubernetes-preserve-unknown-fields: true
    type: object
    description: this is arbitrary JSON
```

With this, only `object` type values are allowed.

Pruning is enabled again for each specified property (or `additionalProperties`):

```
type: object
properties:
  json:
    x-kubernetes-preserve-unknown-fields: true
    type: object
    properties:
      spec:
        type: object
        properties:
          foo:
            type: string
          bar:
            type: string
```

With this, the value:

```
json:
  spec:
    foo: abc
    bar: def
    something: x
  status:
    something: x
```

is pruned to:

```
json:
  spec:
```

```
  foo: abc
  bar: def
  status:
    something: x
```

This means that the `something` field in the specified `spec` object is pruned, but everything outside is not.

## **IntOrString**

Nodes in a schema with `x-kubernetes-int-or-string: true` are excluded from rule 1, such that the following is structural:

```
  type: object
  properties:
    foo:
      x-kubernetes-int-or-string: true
```

Also those nodes are partially excluded from rule 3 in the sense that the following two patterns are allowed (exactly those, without variations in order to additional fields):

```
  x-kubernetes-int-or-string: true
  anyOf:
    - type: integer
    - type: string
    ...
```

and

```
  x-kubernetes-int-or-string: true
  allOf:
    - anyOf:
      - type: integer
      - type: string
    - ... # zero or more
  ...
```

With one of those specification, both an integer and a string validate.

In [Validation Schema Publishing](#), `x-kubernetes-int-or-string: true` is unfolded to one of the two patterns shown above.

## **RawExtension**

RawExtensions (as in `runtime.RawExtension` defined in [k8s.io/apimachinery](#)) holds complete Kubernetes objects, i.e. with `apiVersion` and `kind` fields.

It is possible to specify those embedded objects (both completely without constraints or partially specified) by setting `x-kubernetes-embedded-resource: true`. For example:

```
type: object
properties:
  foo:
    x-kubernetes-embedded-resource: true
    x-kubernetes-preserve-unknown-fields: true
```

Here, the field `foo` holds a complete object, e.g.:

```
foo:
  apiVersion: v1
  kind: Pod
  spec:
    ...
```

Because `x-kubernetes-preserve-unknown-fields: true` is specified alongside, nothing is pruned. The use of `x-kubernetes-preserve-unknown-fields: true` is optional though.

With `x-kubernetes-embedded-resource: true`, the `apiVersion`, `kind` and `metadata` are implicitly specified and validated.

## **Serving multiple versions of a CRD**

See [Custom resource definition versioning](#) for more information about serving multiple versions of your CustomResourceDefinition and migrating your objects from one version to another.

## **Advanced topics**

### **Finalizers**

*Finalizers allow controllers to implement asynchronous pre-delete hooks. Custom objects support finalizers just like built-in objects.*

*You can add a finalizer to a custom object like this:*

```
apiVersion: "stable.example.com/v1"
kind: CronTab
metadata:
  finalizers:
    - stable.example.com/finalizer
```

*Identifiers of custom finalizers consist of a domain name, a forward slash and the name of the finalizer. Any controller can add a finalizer to any object's list of finalizers.*

*The first delete request on an object with finalizers sets a value for the `metadata.deletionTimestamp` field but does not delete it. Once this value is set, entries in the `finalizers` list can only be removed. While any finalizers remain it is also impossible to force the deletion of an object.*

*When the `metadata.deletionTimestamp` field is set, controllers watching the object execute any finalizers they handle and remove the finalizer from the list after they are done. It is the responsibility of each controller to remove its finalizer from the list.*

*The value of `metadata.deletionGracePeriodSeconds` controls the interval between polling updates.*

*Once the list of finalizers is empty, meaning all finalizers have been executed, the resource is deleted by Kubernetes.*

## **Validation**

*Custom resources are validated via [OpenAPI v3 schemas](#) and you can add additional validation using [admission webhooks](#).*

*Additionally, the following restrictions are applied to the schema:*

- These fields cannot be set:
  - `definitions`,
  - `dependencies`,
  - `deprecated`,
  - `discriminator`,

- `id`,
- `patternProperties`,
- `readOnly`,
- `writeOnly`,
- `xml`,
- `$ref`.
- The field `uniqueItems` cannot be set to `true`.
- The field `additionalProperties` cannot be set to `false`.
- The field `additionalProperties` is mutually exclusive with `properties`.

The `default` field can be set when the [Defaulting feature](#) is enabled, which is the case with `apiextensions.k8s.io/v1`

`CustomResourceDefinitions`. Defaulting is in GA since 1.17 (beta since 1.16 with the `CustomResourceDefaulting` [feature gate](#) enabled, which is the case automatically for many clusters for beta features).

Refer to the [structural schemas](#) section for other restrictions and `CustomResourceDefinition` features.

The schema is defined in the `CustomResourceDefinition`. In the following example, the `CustomResourceDefinition` applies the following validations on the custom object:

- `spec.cronSpec` must be a string and must be of the form described by the regular expression.
- `spec.replicas` must be an integer and must have a minimum value of 1 and a maximum value of 10.

Save the `CustomResourceDefinition` to `resourcedefinition.yaml`:

```
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  name: crontabs.stable.example.com
spec:
  group: stable.example.com
  versions:
    - name: v1
      served: true
      storage: true
      schema:
        # openAPIV3Schema is the schema for validating
        # custom objects.
        openAPIV3Schema:
          type: object
          properties:
            spec:
              type: object
              properties:
```

```

    cronSpec:
      type: string
      pattern: '^(\d+/*)(/\d+)?(\s+(\d+/*)(/\d+)?){4}$'
    image:
      type: string
    replicas:
      type: integer
      minimum: 1
      maximum: 10
  scope: Namespaced
  names:
    plural: crontabs
    singular: crontab
    kind: CronTab
    shortNames:
      - ct

```

and create it:

```
kubectl apply -f resourcedefinition.yaml
```

A request to create a custom object of kind CronTab is rejected if there are invalid values in its fields. In the following example, the custom object contains fields with invalid values:

- spec.cronSpec does not match the regular expression.
- spec.replicas is greater than 10.

If you save the following YAML to my-crontab.yaml:

```

apiVersion: "stable.example.com/v1"
kind: CronTab
metadata:
  name: my-new-cron-object
spec:
  cronSpec: "* * * *"
  image: my-awesome-cron-image
  replicas: 15

```

and attempt to create it:

```
kubectl apply -f my-crontab.yaml
```

then you get an error:

```
The CronTab "my-new-cron-object" is invalid: []: Invalid
value: map[string]interface {}
{"apiVersion": "stable.example.com/v1", "kind": "CronTab",
"metadata": map[string]interface {}{"name": "my-new-cron-
object", "namespace": "default",
"deletionTimestamp": interface {}(nil),
"deletionGracePeriodSeconds": (*int64)(nil),
"creationTimestamp": "2017-09-05T05:20:07Z",
"uid": "e14d79e7-91f9-11e7-a598-f0761cb232d1",
"clusterName": ""}, "spec": map[string]interface {}
{"cronSpec": "* * * * *", "image": "my-awesome-cron-image",
"replicas": 15}]:
validation failure list:
spec.cronSpec in body should match '^(\d+|*)(/\d+)?(\s+(\d+|
*)/(\d+)?){4}$'
spec.replicas in body should be less than or equal to 10
```

If the fields contain valid values, the object creation request is accepted.

Save the following YAML to `my-crontab.yaml`:

```
apiVersion: "stable.example.com/v1"
kind: CronTab
metadata:
  name: my-new-cron-object
spec:
  cronSpec: "* * * * */5"
  image: my-awesome-cron-image
  replicas: 5
```

And create it:

```
kubectl apply -f my-crontab.yaml
crontab "my-new-cron-object" created
```

## Defaulting

**Note:** To use defaulting, your `CustomResourceDefinition` must use API version `apiextensions.k8s.io/v1`.

Defaulting allows to specify default values in the [OpenAPI v3 validation schema](#):

```

apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  name: crontabs.stable.example.com
spec:
  group: stable.example.com
  versions:
    - name: v1
      served: true
      storage: true
      schema:
        # openAPIV3Schema is the schema for validating
        custom objects.
        openAPIV3Schema:
          type: object
          properties:
            spec:
              type: object
              properties:
                cronSpec:
                  type: string
                  pattern: '^(\d+/\*)|(\d+)?(\s+(\d+/\*)|(\d+/\*){4})$'
                  default: "5 0 * * *"
                image:
                  type: string
                replicas:
                  type: integer
                  minimum: 1
                  maximum: 10
                  default: 1
            scope: Namespaced
            names:
              plural: crontabs
              singular: crontab
              kind: CronTab
              shortNames:
                - ct

```

With this both `cronSpec` and `replicas` are defaulted:

```

apiVersion: "stable.example.com/v1"
kind: CronTab
metadata:
  name: my-new-cron-object
spec:
  image: my-awesome-cron-image

```

leads to

```

apiVersion: "stable.example.com/v1"
kind: CronTab
metadata:
  name: my-new-cron-object
spec:
  cronSpec: "5 0 * * *"
  image: my-awesome-cron-image
  replicas: 1

```

*Defaulting happens on the object*

- in the request to the API server using the request version defaults,
- when reading from etcd using the storage version defaults,
- after mutating admission plugins with non-empty patches using the admission webhook object version defaults.

*Defaults applied when reading data from etcd are not automatically written back to etcd. An update request via the API is required to persist those defaults back into etcd.*

*Default values must be pruned (with the exception of defaults for metadata fields) and must validate against a provided schema.*

*Default values for metadata fields of x-kubernetes-embedded-resources: true nodes (or parts of a default value covering metadata) are not pruned during CustomResourceDefinition creation, but through the pruning step during handling of requests.*

## **Defaulting and Nullable**

**New in 1.20:** null values for fields that either don't specify the nullable flag, or give it a false value, will be pruned before defaulting happens. If a default is present, it will be applied. When nullable is true, null values will be conserved and won't be defaulted.

*For example, given the OpenAPI schema below:*

```

type: object
properties:
  spec:
    type: object
    properties:
      foo:
        type: string
        nullable: false
        default: "default"

```

```
bar:  
  type: string  
  nullable: true  
baz:  
  type: string
```

*creating an object with null values for foo and bar and baz*

```
spec:  
  foo: null  
  bar: null  
  baz: null
```

*leads to*

```
spec:  
  foo: "default"  
  bar: null
```

*with foo pruned and defaulted because the field is non-nullable, bar maintaining the null value due to nullable: true, and baz pruned because the field is non-nullable and has no default.*

## **Publish Validation Schema in OpenAPI v2**

*CustomResourceDefinition [OpenAPI v3 validation schemas](#) which are [structural](#) and [enable pruning](#) are published as part of the [OpenAPI v2 spec](#) from Kubernetes API server.*

*The [kubectl](#) command-line tool consumes the published schema to perform client-side validation (`kubectl create` and `kubectl apply`), schema explanation (`kubectl explain`) on custom resources. The published schema can be consumed for other purposes as well, like client generation or documentation.*

*The OpenAPI v3 validation schema is converted to OpenAPI v2 schema, and show up in definitions and paths fields in the [OpenAPI v2 spec](#).*

*The following modifications are applied during the conversion to keep backwards compatibility with kubectl in previous 1.13 version. These modifications prevent kubectl from being over-strict and rejecting valid OpenAPI schemas that it doesn't understand. The conversion won't modify the validation schema defined in CRD, and therefore won't affect [validation](#) in the API server.*

1. The following fields are removed as they aren't supported by OpenAPI v2 (in future versions OpenAPI v3 will be used without these restrictions)
  - The fields `allOf`, `anyOf`, `oneOf` and `not` are removed
2. If `nullable: true` is set, we drop `type`, `nullable`, `items` and `properties` because OpenAPI v2 is not able to express nullable. To avoid `kubectl` to reject good objects, this is necessary.

## **Additional printer columns**

The `kubectl` tool relies on server-side output formatting. Your cluster's API server decides which columns are shown by the `kubectl get` command. You can customize these columns for a `CustomResourceDefinition`. The following example adds the `Spec`, `Replicas`, and `Age` columns.

Save the `CustomResourceDefinition` to `resourcedefinition.yaml`:

```
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  name: crontabs.stable.example.com
spec:
  group: stable.example.com
  scope: Namespaced
  names:
    plural: crontabs
    singular: crontab
    kind: CronTab
    shortNames:
    - ct
  versions:
  - name: v1
    served: true
    storage: true
    schema:
      openAPIV3Schema:
        type: object
        properties:
          spec:
            type: object
            properties:
              cronSpec:
                type: string
              image:
                type: string
              replicas:
                type: integer
  additionalPrinterColumns:
```

```
- name: Spec
  type: string
  description: The cron spec defining the interval a Cron
Job is run
  jsonPath: .spec.cronSpec
- name: Replicas
  type: integer
  description: The number of jobs launched by the CronJob
  jsonPath: .spec.replicas
- name: Age
  type: date
  jsonPath: .metadata.creationTimestamp
```

Create the CustomResourceDefinition:

```
kubectl apply -f resourcedefinition.yaml
```

Create an instance using the `my-crontab.yaml` from the previous section.

Invoke the server-side printing:

```
kubectl get crontab my-new-cron-object
```

Notice the `NAME`, `SPEC`, `REPLICAS`, and `AGE` columns in the output:

NAME	SPEC	REPLICAS	AGE
<code>my-new-cron-object</code>	<code>* * * * *</code>	<code>1</code>	<code>7s</code>

**Note:** The `NAME` column is implicit and does not need to be defined in the CustomResourceDefinition.

## Priority

Each column includes a `priority` field. Currently, the priority differentiates between columns shown in standard view or wide view (using the `-o wide` flag).

- Columns with priority 0 are shown in standard view.
- Columns with priority greater than 0 are shown only in wide view.

## Type

A column's `type` field can be any of the following (compare [OpenAPI v3 data types](#)):

- `integer` - non-floating-point numbers
- `number` - floating point numbers
- `string` - strings
- `boolean` - true or false
- `date` - rendered differentially as time since this timestamp.

If the value inside a CustomResource does not match the type specified for the column, the value is omitted. Use CustomResource validation to ensure that the value types are correct.

## **Format**

A column's `format` field can be any of the following:

- `int32`
- `int64`
- `float`
- `double`
- `byte`
- `date`
- `date-time`
- `password`

The column's `format` controls the style used when `kubectl` prints the value.

## **Subresources**

Custom resources support `/status` and `/scale` subresources.

The `status` and `scale` subresources can be optionally enabled by defining them in the CustomResourceDefinition.

### **Status subresource**

When the status subresource is enabled, the `/status` subresource for the custom resource is exposed.

- The `status` and the `spec` stanzas are represented by the `.status` and `.spec` JSONPaths respectively inside of a custom resource.

- PUT requests to the /status subresource take a custom resource object and ignore changes to anything except the status stanza.*
- *PUT requests to the /status subresource only validate the status stanza of the custom resource.*
  - *PUT/POST/PATCH requests to the custom resource ignore changes to the status stanza.*
  - *The .metadata.generation value is incremented for all changes, except for changes to .metadata or .status.*
  - *Only the following constructs are allowed at the root of the CRD OpenAPI validation schema:*
    - *description*
    - *example*
    - *exclusiveMaximum*
    - *exclusiveMinimum*
    - *externalDocs*
    - *format*
    - *items*
    - *maximum*
    - *maxItems*
    - *maxLength*
    - *minimum*
    - *minItems*
    - *minLength*
    - *multipleOf*
    - *pattern*
    - *properties*
    - *required*
    - *title*
    - *type*
    - *uniqueItems*

## **Scale subresource**

*When the scale subresource is enabled, the /scale subresource for the custom resource is exposed. The autoscaling/v1.Scale object is sent as the payload for /scale.*

*To enable the scale subresource, the following fields are defined in the CustomResourceDefinition.*

- *specReplicasPath defines the JSONPath inside of a custom resource that corresponds to scale.spec.replicas.*
  - *It is a required value.*

- Only JSONPaths under `.spec` and with the dot notation are allowed.
- If there is no value under the `specReplicasPath` in the custom resource, the `/scale` subresource will return an error on GET.
- `statusReplicasPath` defines the JSONPath inside of a custom resource that corresponds to `scale.status.replicas`.
  - It is a required value.
  - Only JSONPaths under `.status` and with the dot notation are allowed.
  - If there is no value under the `statusReplicasPath` in the custom resource, the status replica value in the `/scale` subresource will default to 0.
- `labelSelectorPath` defines the JSONPath inside of a custom resource that corresponds to `Scale.Status.Selector`.
  - It is an optional value.
  - It must be set to work with HPA.
  - Only JSONPaths under `.status` or `.spec` and with the dot notation are allowed.
  - If there is no value under the `labelSelectorPath` in the custom resource, the status selector value in the `/scale` subresource will default to the empty string.
  - The field pointed by this JSON path must be a string field (not a complex selector struct) which contains a serialized label selector in string form.

In the following example, both `status` and `scale` subresources are enabled.

Save the CustomResourceDefinition to `resourcedefinition.yaml`:

```
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  name: crontabs.stable.example.com
spec:
  group: stable.example.com
  versions:
    - name: v1
      served: true
      storage: true
      schema:
        openAPIV3Schema:
          type: object
          properties:
            spec:
              type: object
```

```

properties:
  cronSpec:
    type: string
  image:
    type: string
  replicas:
    type: integer
  status:
    type: object
    properties:
      replicas:
        type: integer
      labelSelector:
        type: string
  # subresources describes the subresources for custom
  resources.
  subresources:
    # status enables the status subresource.
    status: {}
    # scale enables the scale subresource.
    scale:
      # specReplicasPath defines the JSONPath inside of
      a custom resource that corresponds to Scale.Spec.Replicas.
      specReplicasPath: .spec.replicas
      # statusReplicasPath defines the JSONPath inside
      of a custom resource that corresponds to
      Scale.Status.Replicas.
      statusReplicasPath: .status.replicas
      # labelSelectorPath defines the JSONPath inside of
      a custom resource that corresponds to Scale.Status.Selector.
      labelSelectorPath: .status.labelSelector
  scope: Namespaced
  names:
    plural: crontabs
    singular: crontab
    kind: CronTab
    shortNames:
      - ct

```

And create it:

```
kubectl apply -f resourcedefinition.yaml
```

After the CustomResourceDefinition object has been created, you can create custom objects.

If you save the following YAML to `my-crontab.yaml`:

```
apiVersion: "stable.example.com/v1"
kind: CronTab
metadata:
  name: my-new-cron-object
spec:
  cronSpec: "* * * * */5"
  image: my-awesome-cron-image
  replicas: 3
```

and create it:

```
kubectl apply -f my-crontab.yaml
```

Then new namespaced RESTful API endpoints are created at:

```
/apis/stable.example.com/v1/namespaces/*/crontabs/status
```

and

```
/apis/stable.example.com/v1/namespaces/*/crontabs/scale
```

A custom resource can be scaled using the `kubectl scale` command. For example, the following command sets `.spec.replicas` of the custom resource created above to 5:

```
kubectl scale --replicas=5 crontabs/my-new-cron-object
crontabs "my-new-cron-object" scaled

kubectl get crontabs my-new-cron-object -o jsonpath='{.spec.r
eplicas}'
5
```

You can use a [PodDisruptionBudget](#) to protect custom resources that have the `scale` subresource enabled.

## Categories

**FEATURE STATE:** Kubernetes v1.10 [beta]

`Categories` is a list of grouped resources the custom resource belongs to (eg. `all`). You can use `kubectl get <category-name>` to list the resources belonging to the category.

The following example adds `all` in the list of categories in the `CustomResourceDefinition` and illustrates how to output the custom resource using `kubectl get all`.

Save the following `CustomResourceDefinition` to `resourcedefinition.yaml`:

```
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  name: crontabs.stable.example.com
spec:
  group: stable.example.com
  versions:
    - name: v1
      served: true
      storage: true
      schema:
        openAPIV3Schema:
          type: object
          properties:
            spec:
              type: object
              properties:
                cronSpec:
                  type: string
                image:
                  type: string
                replicas:
                  type: integer
  scope: Namespaced
  names:
    plural: crontabs
    singular: crontab
    kind: CronTab
    shortNames:
      - ct
    # categories is a list of grouped resources the custom
    # resource belongs to.
    categories:
      - all
```

and create it:

```
kubectl apply -f resourcedefinition.yaml
```

After the `CustomResourceDefinition` object has been created, you can create custom objects.

Save the following YAML to `my-crontab.yaml`:

```
apiVersion: "stable.example.com/v1"
kind: CronTab
metadata:
  name: my-new-cron-object
spec:
  cronSpec: "* * * * */5"
  image: my-awesome-cron-image
```

and create it:

```
kubectl apply -f my-crontab.yaml
```

You can specify the category when using `kubectl get`:

```
kubectl get all
```

and it will include the custom resources of kind `CronTab`:

NAME	AGE
crontabs/my-new-cron-object	3s

## What's next

- Read about [custom resources](#).
- See [CustomResourceDefinition](#).
- Serve [multiple versions](#) of a CustomResourceDefinition.

## 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](#).

Last modified November 19, 2020 at 10:31 AM PST: [Add section for defaulting and nullable \(3dc68c924\)](#)

[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Create a CustomResourceDefinition](#)
- [Create custom objects](#)
- [Delete a CustomResourceDefinition](#)
- [Specifying a structural schema](#)
  - [Field pruning](#)
  - [IntOrString](#)
  - [RawExtension](#)
- [Serving multiple versions of a CRD](#)
- [Advanced topics](#)
  - [Finalizers](#)
  - [Validation](#)
  - [Defaulting](#)
  - [Publish Validation Schema in OpenAPI v2](#)
  - [Additional printer columns](#)
  - [Subresources](#)
  - [Categories](#)
- [What's next](#)

# Versions in CustomResourceDefinitions

This page explains how to add versioning information to [CustomResourceDefinitions](#), to indicate the stability level of your CustomResourceDefinitions or advance your API to a new version with conversion between API representations. It also describes how to upgrade an object from one version to another.

## Before you begin

You need to have a Kubernetes cluster, and the `kubectl` command-line tool must be configured to communicate with your cluster. If you do not already have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:

- [Katacoda](#)
- [Play with Kubernetes](#)

You should have a initial understanding of [custom resources](#).

Your Kubernetes server must be at or later than version v1.16. To check the version, enter `kubectl version`.

## Overview

*The CustomResourceDefinition API provides a workflow for introducing and upgrading to new versions of a CustomResourceDefinition.*

*When a CustomResourceDefinition is created, the first version is set in the CustomResourceDefinition spec.versions list to an appropriate stability level and a version number. For example v1beta1 would indicate that the first version is not yet stable. All custom resource objects will initially be stored at this version.*

*Once the CustomResourceDefinition is created, clients may begin using the v1beta1 API.*

*Later it might be necessary to add new version such as v1.*

*Adding a new version:*

1. *Pick a conversion strategy. Since custom resource objects need to be able to be served at both versions, that means they will sometimes be served at a different version than their storage version. In order for this to be possible, the custom resource objects must sometimes be converted between the version they are stored at and the version they are served at. If the conversion involves schema changes and requires custom logic, a conversion webhook should be used. If there are no schema changes, the default None conversion strategy may be used and only the apiVersion field will be modified when serving different versions.*
2. *If using conversion webhooks, create and deploy the conversion webhook. See the [Webhook conversion](#) for more details.*
3. *Update the CustomResourceDefinition to include the new version in the spec.versions list with served:true. Also, set spec.conversion field to the selected conversion strategy. If using a conversion webhook, configure spec.conversion.webhookClientConfig field to call the webhook.*

*Once the new version is added, clients may incrementally migrate to the new version. It is perfectly safe for some clients to use the old version while others use the new version.*

*Migrate stored objects to the new version:*

1. *See the [upgrade existing objects to a new stored version](#) section.*

*It is safe for clients to use both the old and new version before, during and after upgrading the objects to a new stored version.*

*Removing an old version:*

1. Ensure all clients are fully migrated to the new version. The kube-apiserver logs can be reviewed to help identify any clients that are still accessing via the old version.
2. Set `served` to `false` for the old version in the `spec.versions` list. If any clients are still unexpectedly using the old version they may begin reporting errors attempting to access the custom resource objects at the old version. If this occurs, switch back to using `served:true` on the old version, migrate the remaining clients to the new version and repeat this step.
3. Ensure the [upgrade of existing objects to the new stored version](#) step has been completed.
  1. Verify that the `stored` is set to `true` for the new version in the `spec.versions` list in the CustomResourceDefinition.
  2. Verify that the old version is no longer listed in the CustomResourceDefinition `status.storedVersions`.
4. Remove the old version from the CustomResourceDefinition `spec.versions` list.
5. Drop conversion support for the old version in conversion webhooks.

## Specify multiple versions

The CustomResourceDefinition API `versions` field can be used to support multiple versions of custom resources that you have developed. Versions can have different schemas, and conversion webhooks can convert custom resources between versions. Webhook conversions should follow the [Kubernetes API conventions](#) wherever applicable. Specifically, See the [API change documentation](#) for a set of useful gotchas and suggestions.

**Note:** In `apiextensions.k8s.io/v1beta1`, there was a `version` field instead of `versions`. The `version` field is deprecated and optional, but if it is not empty, it must match the first item in the `versions` field.

This example shows a CustomResourceDefinition with two versions. For the first example, the assumption is all versions share the same schema with no conversion between them. The comments in the YAML provide more context.

- [apiextensions.k8s.io/v1](#)
- [apiextensions.k8s.io/v1beta1](#)

```
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  # name must match the spec fields below, and be in the
  form: <plural>.<group>
  name: crontabs.example.com
```

```

spec:
  # group name to use for REST API: /apis/<group>/<version>
  group: example.com
  # list of versions supported by this
  CustomResourceDefinition
    versions:
      - name: v1beta1
        # Each version can be enabled/disabled by Served flag.
        served: true
        # One and only one version must be marked as the storage
        # version.
        storage: true
        # A schema is required
        schema:
          openAPIV3Schema:
            type: object
            properties:
              host:
                type: string
              port:
                type: string
      - name: v1
        served: true
        storage: false
        schema:
          openAPIV3Schema:
            type: object
            properties:
              host:
                type: string
              port:
                type: string
    # The conversion section is introduced in Kubernetes 1.13+
    # with a default value of
    # None conversion (strategy sub-field set to None).
    conversion:
      # None conversion assumes the same schema for all
      # versions and only sets the apiVersion
      # field of custom resources to the proper value
      strategy: None
    # either Namespaced or Cluster
    scope: Namespaced
    names:
      # plural name to be used in the URL: /apis/<group>/
      <version>/<plural>
      plural: crontabs
      # singular name to be used as an alias on the CLI and
      # for display
      singular: crontab
      # kind is normally the CamelCased singular type. Your
      # resource manifests use this.
      kind: CronTab

```

```

# shortNames allow shorter string to match your resource
on the CLI
shortNames:
- ct

# Deprecated in v1.16 in favor of apiextensions.k8s.io/v1
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
# name must match the spec fields below, and be in the
form: <plural>.<group>
name: crontabs.example.com
spec:
# group name to use for REST API: /apis/<group>/<version>
group: example.com
# list of versions supported by this
CustomResourceDefinition
versions:
- name: v1beta1
# Each version can be enabled/disabled by Served flag.
served: true
# One and only one version must be marked as the storage
version.
storage: true
- name: v1
served: true
storage: false
validation:
openAPIV3Schema:
type: object
properties:
host:
type: string
port:
type: string
# The conversion section is introduced in Kubernetes 1.13+
with a default value of
# None conversion (strategy sub-field set to None).
conversion:
# None conversion assumes the same schema for all
versions and only sets the apiVersion
# field of custom resources to the proper value
strategy: None
# either Namespaced or Cluster
scope: Namespaced
names:
# plural name to be used in the URL: /apis/<group>/
<version>/<plural>
plural: crontabs
# singular name to be used as an alias on the CLI and
for display
singular: crontab

```

```
# kind is normally the CamelCased singular type. Your
resource manifests use this.
kind: CronTab
# shortNames allow shorter string to match your resource
on the CLI
shortNames:
- ct
```

You can save the CustomResourceDefinition in a YAML file, then use `kubectl apply` to create it.

```
kubectl apply -f my-versioned-crontab.yaml
```

After creation, the API server starts to serve each enabled version at an HTTP REST endpoint. In the above example, the API versions are available at `/apis/example.com/v1beta1` and `/apis/example.com/v1`.

## Version priority

Regardless of the order in which versions are defined in a CustomResourceDefinition, the version with the highest priority is used by `kubectl` as the default version to access objects. The priority is determined by parsing the name field to determine the version number, the stability (GA, Beta, or Alpha), and the sequence within that stability level.

The algorithm used for sorting the versions is designed to sort versions in the same way that the Kubernetes project sorts Kubernetes versions. Versions start with a `v` followed by a number, an optional `beta` or `alpha` designation, and optional additional numeric versioning information. Broadly, a version string might look like `v2` or `v2beta1`. Versions are sorted using the following algorithm:

- Entries that follow Kubernetes version patterns are sorted before those that do not.
- For entries that follow Kubernetes version patterns, the numeric portions of the version string is sorted largest to smallest.
- If the strings `beta` or `alpha` follow the first numeric portion, they sorted in that order, after the equivalent string without the `beta` or `alpha` suffix (which is presumed to be the GA version).
- If another number follows the `beta`, or `alpha`, those numbers are also sorted from largest to smallest.
- Strings that don't fit the above format are sorted alphabetically and the numeric portions are not treated specially. Notice that in the example below, `foo1` is sorted above `foo10`. This is different from the sorting of the numeric portion of entries that do follow the Kubernetes version patterns.

*This might make sense if you look at the following sorted version list:*

```
- v10
- v2
- v1
- v11beta2
- v10beta3
- v3beta1
- v12alpha1
- v11alpha2
- fool
- fool0
```

*For the example in [Specify multiple versions](#), the version sort order is v1, followed by v1beta1. This causes the kubectl command to use v1 as the default version unless the provided object specifies the version.*

## **Version deprecation**

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

*Starting in v1.19, a CustomResourceDefinition can indicate a particular version of the resource it defines is deprecated. When API requests to a deprecated version of that resource are made, a warning message is returned in the API response as a header. The warning message for each deprecated version of the resource can be customized if desired.*

*A customized warning message should indicate the deprecated API group, version, and kind, and should indicate what API group, version, and kind should be used instead, if applicable.*

- [apiextensions.k8s.io/v1](#)
- [apiextensions.k8s.io/v1beta1](#)

```
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
  name: crontabs.example.com
spec:
  group: example.com
  names:
    plural: crontabs
    singular: crontab
    kind: CronTab
  scope: Namespaced
  versions:
    - name: v1alpha1
      served: true
      # This indicates the v1alpha1 version of the custom
```

```
resource is deprecated.
# API requests to this version receive a warning header
in the server response.
  deprecated: true
    # This overrides the default warning returned to API
    clients making v1alpha1 API requests.
    deprecationWarning: "example.com/v1alpha1 CronTab is
deprecated; see http://example.com/v1alpha1-v1 for
instructions to migrate to example.com/v1 CronTab"
    schema: ...
  - name: v1beta1
    served: true
      # This indicates the v1beta1 version of the custom
      resource is deprecated.
      # API requests to this version receive a warning header
      in the server response.
      # A default warning message is returned for this version.
      deprecated: true
      schema: ...
  - name: v1
    served: true
    storage: true
    schema: ...
```

```
# Deprecated in v1.16 in favor of apiextensions.k8s.io/v1
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
  name: crontabs.example.com
spec:
  group: example.com
  names:
    plural: crontabs
    singular: crontab
    kind: CronTab
  scope: Namespaced
  validation: ...
  versions:
  - name: v1alpha1
    served: true
      # This indicates the v1alpha1 version of the custom
      resource is deprecated.
      # API requests to this version receive a warning header
      in the server response.
      deprecated: true
      # This overrides the default warning returned to API
      clients making v1alpha1 API requests.
      deprecationWarning: "example.com/v1alpha1 CronTab is
deprecated; see http://example.com/v1alpha1-v1 for
instructions to migrate to example.com/v1 CronTab"
  - name: v1beta1
    served: true
```

```
# This indicates the v1beta1 version of the custom
resource is deprecated.
# API requests to this version receive a warning header
in the server response.
# A default warning message is returned for this version.
deprecated: true
- name: v1
  served: true
  storage: true
```

## Webhook conversion

**FEATURE STATE:** Kubernetes v1.16 [stable]

**Note:** Webhook conversion is available as beta since 1.15, and as alpha since Kubernetes 1.13. The `CustomResourceWebhookConversion` feature must be enabled, which is the case automatically for many clusters for beta features. Please refer to the [feature gate](#) documentation for more information.

The above example has a `None` conversion between versions which only sets the `apiVersion` field on conversion and does not change the rest of the object. The API server also supports webhook conversions that call an external service in case a conversion is required. For example when:

- custom resource is requested in a different version than stored version.
- Watch is created in one version but the changed object is stored in another version.
- custom resource PUT request is in a different version than storage version.

To cover all of these cases and to optimize conversion by the API server, the conversion requests may contain multiple objects in order to minimize the external calls. The webhook should perform these conversions independently.

## Write a conversion webhook server

Please refer to the implementation of the [custom resource conversion webhook server](#) that is validated in a Kubernetes e2e test. The webhook handles the `ConversionReview` requests sent by the API servers, and sends back conversion results wrapped in `ConversionResponse`. Note that the request contains a list of custom resources that need to be converted independently without changing the order of objects. The example server is organized in a way to be reused for other conversions. Most of the common code are located in the [framework](#)

[file](#) that leaves only [one function](#) to be implemented for different conversions.

**Note:** The example conversion webhook server leaves the `ClientAuth` field [empty](#), which defaults to `NoClientCert`. This means that the webhook server does not authenticate the identity of the clients, supposedly API servers. If you need mutual TLS or other ways to authenticate the clients, see how to [authenticate API servers](#).

### **Permissible mutations**

A conversion webhook must not mutate anything inside of `metadata` of the converted object other than `labels` and `annotations`. Attempted changes to `name`, `UID` and `namespace` are rejected and fail the request which caused the conversion. All other changes are just ignored.

### **Deploy the conversion webhook service**

Documentation for deploying the conversion webhook is the same as for the [admission webhook example service](#). The assumption for next sections is that the conversion webhook server is deployed to a service named `example-conversion-webhook-server` in default namespace and serving traffic on path `/crdconvert`.

**Note:** When the webhook server is deployed into the Kubernetes cluster as a service, it has to be exposed via a service on port 443 (The server itself can have an arbitrary port but the service object should map it to port 443). The communication between the API server and the webhook service may fail if a different port is used for the service.

### **Configure CustomResourceDefinition to use conversion webhooks**

The `None` conversion example can be extended to use the conversion webhook by modifying `conversion` section of the `spec`:

- [apiextensions.k8s.io/v1](#)
- [apiextensions.k8s.io/v1beta1](#)

```
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  # name must match the spec fields below, and be in the
```

```

form: <plural>.<group>
  name: crontabs.example.com
spec:
  # group name to use for REST API: /apis/<group>/<version>
  group: example.com
  # list of versions supported by this
CustomResourceDefinition
  versions:
    - name: v1beta1
      # Each version can be enabled/disabled by Served flag.
      served: true
      # One and only one version must be marked as the storage
      version.
      storage: true
      # Each version can define it's own schema when there is
      no top-level
      # schema is defined.
      schema:
        openAPI3Schema:
          type: object
          properties:
            hostPort:
              type: string
    - name: v1
      served: true
      storage: false
      schema:
        openAPI3Schema:
          type: object
          properties:
            host:
              type: string
            port:
              type: string
  conversion:
    # a Webhook strategy instruct API server to call an
    external webhook for any conversion between custom resources.
    strategy: Webhook
    # webhook is required when strategy is `Webhook` and it
    configures the webhook endpoint to be called by API server.
    webhook:
      # conversionReviewVersions indicates what
      ConversionReview versions are understood/preferred by the
      webhook.
      # The first version in the list understood by the API
      server is sent to the webhook.
      # The webhook must respond with a ConversionReview
      object in the same version it received.
      conversionReviewVersions: ["v1", "v1beta1"]
      clientConfig:
        service:
          namespace: default

```

```

        name: example-conversion-webhook-server
        path: /crdconvert
        caBundle: "Ci0tLS0tQk...<base64-encoded PEM
bundle>...tLS0K"
    # either Namespaced or Cluster
    scope: Namespaced
    names:
        # plural name to be used in the URL: /apis/<group>/
<version>/<plural>
        plural: crontabs
        # singular name to be used as an alias on the CLI and
for display
        singular: crontab
        # kind is normally the CamelCased singular type. Your
resource manifests use this.
        kind: CronTab
        # shortNames allow shorter string to match your resource
on the CLI
        shortNames:
            - ct

# Deprecated in v1.16 in favor of apiextensions.k8s.io/v1
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
    # name must match the spec fields below, and be in the
form: <plural>.<group>
    name: crontabs.example.com
spec:
    # group name to use for REST API: /apis/<group>/<version>
    group: example.com
    # prunes object fields that are not specified in OpenAPI
schemas below.
    preserveUnknownFields: false
    # list of versions supported by this
CustomResourceDefinition
    versions:
        - name: v1beta1
            # Each version can be enabled/disabled by Served flag.
            served: true
            # One and only one version must be marked as the storage
version.
            storage: true
            # Each version can define it's own schema when there is
no top-level
            # schema is defined.
            schema:
                openAPIV3Schema:
                    type: object
                    properties:
                        hostPort:
                            type: string

```

```

- name: v1
  served: true
  storage: false
  schema:
    openAPIV3Schema:
      type: object
      properties:
        host:
          type: string
        port:
          type: string
  conversion:
    # a Webhook strategy instruct API server to call an
    # external webhook for any conversion between custom resources.
    strategy: Webhook
    # webhookClientConfig is required when strategy is
    `Webhook` and it configures the webhook endpoint to be
    called by API server.
    webhookClientConfig:
      service:
        namespace: default
        name: example-conversion-webhook-server
        path: /crdconvert
        caBundle: "Ci0tLS0tQk...<base64-encoded PEM
bundle>...tLS0K"
      # either Namespaced or Cluster
      scope: Namespaced
      names:
        # plural name to be used in the URL: /apis/<group>/
        <version>/<plural>
        plural: crontabs
        # singular name to be used as an alias on the CLI and
        # for display
        singular: crontab
        # kind is normally the CamelCased singular type. Your
        # resource manifests use this.
        kind: CronTab
        # shortNames allow shorter string to match your resource
        # on the CLI
        shortNames:
        - ct

```

You can save the CustomResourceDefinition in a YAML file, then use `kubectl apply` to apply it.

```
kubectl apply -f my-versioned-crontab-with-conversion.yaml
```

Make sure the conversion service is up and running before applying new changes.

## Contacting the webhook

Once the API server has determined a request should be sent to a conversion webhook, it needs to know how to contact the webhook. This is specified in the `webhookClientConfig` stanza of the webhook configuration.

Conversion 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 (i.e., `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 (for example "`user:password@`") is not allowed. Fragments ("`#...`") and query parameters ("`?...`") are also not allowed.

Here is an example of a conversion webhook configured to call a URL (and expects the TLS certificate to be verified using system trust roots, so does not specify a `caBundle`):

- [`apiextensions.k8s.io/v1`](#)
- [`apiextensions.k8s.io/v1beta1`](#)

```
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
```

```
...
spec:
```

```

conversion:
  strategy: Webhook
  webhook:
    clientConfig:
      url: "https://my-webhook.example.com:9443/my-webhook-
path"
...
# Deprecated in v1.16 in favor of apiextensions.k8s.io/v1
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
...
spec:
...
conversion:
  strategy: Webhook
  webhookClientConfig:
    url: "https://my-webhook.example.com:9443/my-webhook-
path"
...

```

## Service Reference

The `service` stanza inside `webhookClientConfig` is a reference to the service for a conversion 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 webhook that is 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.

- [apiextensions.k8s.io/v1](#)
- [apiextensions.k8s.io/v1beta1](#)

```

apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
...
spec:
...
conversion:
  strategy: Webhook
  webhook:
    clientConfig:
      service:
        namespace: my-service-namespace
        name: my-service-name
        path: /my-path

```

```

    port: 1234
    caBundle: "Ci0tLS0tQk...<base64-encoded PEM
bundle>...tLS0K"
...
# Deprecated in v1.16 in favor of apiextensions.k8s.io/v1
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition

...
spec:
  ...
  conversion:
    strategy: Webhook
    webhookClientConfig:
      service:
        namespace: my-service-namespace
        name: my-service-name
        path: /my-path
        port: 1234
    caBundle: "Ci0tLS0tQk...<base64-encoded PEM
bundle>...tLS0K"
...

```

## Webhook request and response

### Request

Webhooks are sent a *POST* request, with *Content-Type: application/json*, with a *ConversionReview API* object in the *apiextensions.k8s.io* API group serialized to JSON as the body.

Webhooks can specify what versions of *ConversionReview* objects they accept with the *conversionReviewVersions* field in their *CustomResourceDefinition*:

- [apiextensions.k8s.io/v1](#)
- [apiextensions.k8s.io/v1beta1](#)

```

apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition

...
spec:
  ...
  conversion:
    strategy: Webhook
    webhook:
      conversionReviewVersions: ["v1", "v1beta1"]
...

```

`conversionReviewVersions` is a required field when creating `apiextensions.k8s.io/v1` custom resource definitions. Webhooks are required to support at least one `ConversionReview` version understood by the current and previous API server.

```
# Deprecated in v1.16 in favor of apiextensions.k8s.io/v1
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition

...
spec:
  ...
    conversion:
      strategy: Webhook
      conversionReviewVersions: ["v1", "v1beta1"]
  ...
  
```

If no `conversionReviewVersions` are specified, the default when creating `apiextensions.k8s.io/v1beta1` custom resource definitions is `v1beta1`.

API servers send the first `ConversionReview` version in the `conversionReviewVersions` list they support. If none of the versions in the list are supported by the API server, the custom resource definition will not be allowed to be created. If an API server encounters a conversion webhook configuration that was previously created and does not support any of the `ConversionReview` versions the API server knows how to send, attempts to call to the webhook will fail.

This example shows the data contained in an `ConversionReview` object for a request to convert `CronTab` objects to `example.com/v1`:

- [apiextensions.k8s.io/v1](#)
- [apiextensions.k8s.io/v1beta1](#)

```
{
  "apiVersion": "apiextensions.k8s.io/v1",
  "kind": "ConversionReview",
  "request": {
    # Random uid uniquely identifying this conversion call
    "uid": "705ab4f5-6393-11e8-b7cc-42010a800002",

    # The API group and version the objects should be
    converted to
    "desiredAPIVersion": "example.com/v1",

    # The list of objects to convert.
    # May contain one or more objects, in one or more
    versions.
    "objects": [
      {
        "kind": "CronTab",
      }
    ]
  }
}
```

```

    "apiVersion": "example.com/v1beta1",
    "metadata": {
        "creationTimestamp": "2019-09-04T14:03:02Z",
        "name": "local-crontab",
        "namespace": "default",
        "resourceVersion": "143",
        "uid": "3415a7fc-162b-4300-b5da-fd6083580d66"
    },
    "hostPort": "localhost:1234"
},
{
    "kind": "CronTab",
    "apiVersion": "example.com/v1beta1",
    "metadata": {
        "creationTimestamp": "2019-09-03T13:02:01Z",
        "name": "remote-crontab",
        "resourceVersion": "12893",
        "uid": "359a83ec-b575-460d-b553-d859cedde8a0"
    },
    "hostPort": "example.com:2345"
}
]
}
}

{
    # Deprecated in v1.16 in favor of apiextensions.k8s.io/v1
    "apiVersion": "apiextensions.k8s.io/v1beta1",
    "kind": "ConversionReview",
    "request": {
        # Random uid uniquely identifying this conversion call
        "uid": "705ab4f5-6393-11e8-b7cc-42010a800002",
        # The API group and version the objects should be
        converted to
        "desiredAPIVersion": "example.com/v1",
        # The list of objects to convert.
        # May contain one or more objects, in one or more
        versions.
        "objects": [
            {
                "kind": "CronTab",
                "apiVersion": "example.com/v1beta1",
                "metadata": {
                    "creationTimestamp": "2019-09-04T14:03:02Z",
                    "name": "local-crontab",
                    "namespace": "default",
                    "resourceVersion": "143",
                    "uid": "3415a7fc-162b-4300-b5da-fd6083580d66"
                },
                "hostPort": "localhost:1234"
            }
        ]
    }
}

```

```

},
{
  "kind": "CronTab",
  "apiVersion": "example.com/v1beta1",
  "metadata": {
    "creationTimestamp": "2019-09-03T13:02:01Z",
    "name": "remote-crontab",
    "resourceVersion": "12893",
    "uid": "359a83ec-b575-460d-b553-d859cedde8a0"
  },
  "hostPort": "example.com:2345"
}
]
}
}

```

## Response

*Webhooks respond with a 200 HTTP status code, Content-Type: application/json, and a body containing a ConversionReview object (in the same version they were sent), with the response stanza populated, serialized to JSON.*

*If conversion succeeds, a webhook should return a response stanza containing the following fields:*

- uid, copied from the request.uid sent to the webhook
- result, set to {"status": "Success"}
- convertedObjects, containing all of the objects from request.objects, converted to request.desiredVersion

*Example of a minimal successful response from a webhook:*

- [apiextensions.k8s.io/v1](#)
- [apiextensions.k8s.io/v1beta1](#)

```

{
  "apiVersion": "apiextensions.k8s.io/v1",
  "kind": "ConversionReview",
  "response": {
    # must match <request.uid>
    "uid": "705ab4f5-6393-11e8-b7cc-42010a800002",
    "result": {
      "status": "Success"
    },
    # Objects must match the order of request.objects, and
    # have apiVersion set to <request.desiredAPIVersion>.
    # kind, metadata.uid, metadata.name, and
    # metadata.namespace fields must not be changed by the webhook.
    # metadata.labels and metadata.annotations fields may be
  }
}

```

```

changed by the webhook.
# All other changes to metadata fields by the webhook
are ignored.
"convertedObjects": [
{
  "kind": "CronTab",
  "apiVersion": "example.com/v1",
  "metadata": {
    "creationTimestamp": "2019-09-04T14:03:02Z",
    "name": "local-crontab",
    "namespace": "default",
    "resourceVersion": "143",
    "uid": "3415a7fc-162b-4300-b5da-fd6083580d66"
  },
  "host": "localhost",
  "port": "1234"
},
{
  "kind": "CronTab",
  "apiVersion": "example.com/v1",
  "metadata": {
    "creationTimestamp": "2019-09-03T13:02:01Z",
    "name": "remote-crontab",
    "resourceVersion": "12893",
    "uid": "359a83ec-b575-460d-b553-d859cedde8a0"
  },
  "host": "example.com",
  "port": "2345"
}
]
}

{
  # Deprecated in v1.16 in favor of apiextensions.k8s.io/v1
  "apiVersion": "apiextensions.k8s.io/v1beta1",
  "kind": "ConversionReview",
  "response": {
    # must match <request.uid>
    "uid": "705ab4f5-6393-11e8-b7cc-42010a800002",
    "result": {
      "status": "Failed"
    },
    # Objects must match the order of request.objects, and
    # have apiVersion set to <request.desiredAPIVersion>.
    # kind, metadata.uid, metadata.name, and
    # metadata.namespace fields must not be changed by the webhook.
    # metadata.labels and metadata.annotations fields may be
    # changed by the webhook.
    # All other changes to metadata fields by the webhook
    # are ignored.
    "convertedObjects": [

```

```

{
  "kind": "CronTab",
  "apiVersion": "example.com/v1",
  "metadata": {
    "creationTimestamp": "2019-09-04T14:03:02Z",
    "name": "local-crontab",
    "namespace": "default",
    "resourceVersion": "143",
    "uid": "3415a7fc-162b-4300-b5da-fd6083580d66"
  },
  "host": "localhost",
  "port": "1234"
},
{
  "kind": "CronTab",
  "apiVersion": "example.com/v1",
  "metadata": {
    "creationTimestamp": "2019-09-03T13:02:01Z",
    "name": "remote-crontab",
    "resourceVersion": "12893",
    "uid": "359a83ec-b575-460d-b553-d859cedde8a0"
  },
  "host": "example.com",
  "port": "2345"
}
]
}

```

If conversion fails, a webhook should return a `response` stanza containing the following fields:

- `uid`, copied from the `request.uid` sent to the webhook
- `result`, set to `{"status": "Failed"}`

**Warning:** Failing conversion can disrupt read and write access to the custom resources, including the ability to update or delete the resources. Conversion failures should be avoided whenever possible, and should not be used to enforce validation constraints (use validation schemas or webhook admission instead).

Example of a response from a webhook indicating a conversion request failed, with an optional message:

- [apiextensions.k8s.io/v1](#)
- [apiextensions.k8s.io/v1beta1](#)

```

{
  "apiVersion": "apiextensions.k8s.io/v1",
  "kind": "ConversionReview",

```

```

"response": {
    "uid": "<value from request.uid>",
    "result": {
        "status": "Failed",
        "message": "hostPort could not be parsed into a
separate host and port"
    }
}
}

{
# Deprecated in v1.16 in favor of apiextensions.k8s.io/v1
"apiVersion": "apiextensions.k8s.io/v1beta1",
"kind": "ConversionReview",
"response": {
    "uid": "<value from request.uid>",
    "result": {
        "status": "Failed",
        "message": "hostPort could not be parsed into a
separate host and port"
    }
}
}

```

## ***Writing, reading, and updating versioned CustomResourceDefinition objects***

*When an object is written, it is persisted at the version designated as the storage version at the time of the write. If the storage version changes, existing objects are never converted automatically. However, newly-created or updated objects are written at the new storage version. It is possible for an object to have been written at a version that is no longer served.*

*When you read an object, you specify the version as part of the path. If you specify a version that is different from the object's persisted version, Kubernetes returns the object to you at the version you requested, but the persisted object is neither changed on disk, nor converted in any way (other than changing the apiVersion string) while serving the request. You can request an object at any version that is currently served.*

*If you update an existing object, it is rewritten at the version that is currently the storage version. This is the only way that objects can change from one version to another.*

*To illustrate this, consider the following hypothetical series of events:*

1. The storage version is `v1beta1`. You create an object. It is persisted in storage at version `v1beta1`
2. You add version `v1` to your CustomResourceDefinition and designate it as the storage version.
3. You read your object at version `v1beta1`, then you read the object again at version `v1`. Both returned objects are identical except for the `apiVersion` field.
4. You create a new object. It is persisted in storage at version `v1`. You now have two objects, one of which is at `v1beta1`, and the other of which is at `v1`.
5. You update the first object. It is now persisted at version `v1` since that is the current storage version.

## **Previous storage versions**

The API server records each version which has ever been marked as the storage version in the status field `storedVersions`. Objects may have been persisted at any version that has ever been designated as a storage version. No objects can exist in storage at a version that has never been a storage version.

## **Upgrade existing objects to a new stored version**

When deprecating versions and dropping support, select a storage upgrade procedure.

### *Option 1: Use the Storage Version Migrator*

1. Run the [storage Version migrator](#)
2. Remove the old version from the CustomResourceDefinition `status.storedVersions` field.

### *Option 2: Manually upgrade the existing objects to a new stored version*

The following is an example procedure to upgrade from `v1beta1` to `v1`.

1. Set `v1` as the storage in the CustomResourceDefinition file and apply it using kubectl. The `storedVersions` is now `v1beta1, v1`.
2. Write an upgrade procedure to list all existing objects and write them with the same content. This forces the backend to write objects in the current storage version, which is `v1`.
3. Remove `v1beta1` from the CustomResourceDefinition `status.storedVersions` field.

# 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](#).

Last modified August 07, 2020 at 8:40 PM PST: [Tune links in tasks section \(2/2\) \(92ae1a9cf\)](#)

[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Overview](#)
- [Specify multiple versions](#)
  - [Version priority](#)
  - [Version deprecation](#)
- [Webhook conversion](#)
  - [Write a conversion webhook server](#)
  - [Deploy the conversion webhook service](#)
  - [Configure CustomResourceDefinition to use conversion webhooks](#)
  - [Contacting the webhook](#)
  - [URL](#)
  - [Service Reference](#)
- [Webhook request and response](#)
  - [Request](#)
  - [Response](#)
- [Writing, reading, and updating versioned CustomResourceDefinition objects](#)
  - [Previous storage versions](#)
- [Upgrade existing objects to a new stored version](#)

# Set up an Extension API Server

Setting up an extension API server to work with the aggregation layer allows the Kubernetes apiserver to be extended with additional APIs, which are not part of the core Kubernetes APIs.

## Before you begin

You need to have a Kubernetes cluster, and the kubectl command-line tool must be configured to communicate with your cluster. If you do not already have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:

- [Katacoda](#)
- [Play with Kubernetes](#)

To check the version, enter `kubectl version`.

- You must [configure the aggregation layer](#) and enable the `apiserver` flags.

## Setup an extension api-server to work with the aggregation layer

The following steps describe how to set up an extension-apiserver at a high level. These steps apply regardless if you're using YAML configs or using APIs. An attempt is made to specifically identify any differences between the two. For a concrete example of how they can be implemented using YAML configs, you can look at the [sample-apiserver](#) in the Kubernetes repo.

Alternatively, you can use an existing 3rd party solution, such as [apiserver-builder](#), which should generate a skeleton and automate all of the following steps for you.

1. Make sure the APIService API is enabled (check `--runtime-config`). It should be on by default, unless it's been deliberately turned off in your cluster.
2. You may need to make an RBAC rule allowing you to add APIService objects, or get your cluster administrator to make one. (Since API extensions affect the entire cluster, it is not recommended to do testing/development/debug of an API extension in a live cluster.)
3. Create the Kubernetes namespace you want to run your extension api-service in.
4. Create/get a CA cert to be used to sign the server cert the extension api-server uses for HTTPS.
5. Create a server cert/key for the api-server to use for HTTPS. This cert should be signed by the above CA. It should also have a CN of the Kube DNS name. This is derived from the Kubernetes service and be of the form `<service name>.<service name namespace>.svc`
6. Create a Kubernetes secret with the server cert/key in your namespace.
7. Create a Kubernetes deployment for the extension api-server and make sure you are loading the secret as a volume. It should contain a reference to a working image of your extension api-server. The deployment should also be in your namespace.
8. Make sure that your extension-apiserver loads those certs from that volume and that they are used in the HTTPS handshake.
9. Create a Kubernetes service account in your namespace.
10. Create a Kubernetes cluster role for the operations you want to allow on your resources.
11. Create a Kubernetes cluster role binding from the service account in your namespace to the cluster role you just created.

12. Create a Kubernetes cluster role binding from the service account in your namespace to the `system:auth-delegator` cluster role to delegate auth decisions to the Kubernetes core API server.
13. Create a Kubernetes role binding from the service account in your namespace to the `extension-apiserver-authentication-reader` role. This allows your extension api-server to access the `extension-apiserver-authentication` configmap.
14. Create a Kubernetes apiservice. The CA cert above should be base64 encoded, stripped of new lines and used as the `spec.caBundle` in the apiservice. This should not be namespaced. If using the [kube-aggregator API](#), only pass in the PEM encoded CA bundle because the base 64 encoding is done for you.
15. Use `kubectl` to get your resource. When run, `kubectl` should return "No resources found.". This message indicates that everything worked but you currently have no objects of that resource type created.

## What's next

- Walk through the steps to [configure the API aggregation layer](#) and enable the apiserver flags.
- For a high level overview, see [Extending the Kubernetes API with the aggregation layer](#).
- Learn how to [Extend the Kubernetes API using Custom Resource Definitions](#).

## 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](#).

Last modified May 04, 2020 at 6:56 PM PST: [move setup konnectivity svc \(5fe8c3ca5\)](#)

[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Setup an extension api-server to work with the aggregation layer](#)
- [What's next](#)

## Configure Multiple Schedulers

Kubernetes ships with a default scheduler that is described [here](#). If the default scheduler does not suit your needs you can implement your own scheduler. Not just that, you can even run multiple schedulers simultaneously alongside the default scheduler and instruct Kubernetes

*what scheduler to use for each of your pods. Let's learn how to run multiple schedulers in Kubernetes with an example.*

*A detailed description of how to implement a scheduler is outside the scope of this document. Please refer to the kube-scheduler implementation in [pkg/scheduler](#) in the Kubernetes source directory for a canonical example.*

## **Before you begin**

*You need to have a Kubernetes cluster, and the kubectl command-line tool must be configured to communicate with your cluster. If you do not already have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:*

- [Katacoda](#)
- [Play with Kubernetes](#)

*To check the version, enter `kubectl version`.*

## **Package the scheduler**

*Package your scheduler binary into a container image. For the purposes of this example, let's just use the default scheduler (kube-scheduler) as our second scheduler as well. Clone the [Kubernetes source code from GitHub](#) and build the source.*

```
git clone https://github.com/kubernetes/kubernetes.git  
cd kubernetes  
make
```

*Create a container image containing the kube-scheduler binary. Here is the Dockerfile to build the image:*

```
FROM busybox  
ADD ./_output/local/bin/linux/amd64/kube-scheduler /usr/  
local/bin/kube-scheduler
```

*Save the file as Dockerfile, build the image and push it to a registry. This example pushes the image to [Google Container Registry \(GCR\)](#). For more details, please read the GCR [documentation](#).*

```
docker build -t gcr.io/my-gcp-project/my-kube-scheduler:1.0 .  
gcloud docker -- push gcr.io/my-gcp-project/my-kube-  
scheduler:1.0
```

# **Define a Kubernetes Deployment for the scheduler**

Now that we have our scheduler in a container image, we can just create a pod config for it and run it in our Kubernetes cluster. But instead of creating a pod directly in the cluster, let's use a [Deployment](#) for this example. A [Deployment](#) manages a [Replica Set](#) which in turn manages the pods, thereby making the scheduler resilient to failures. Here is the deployment config. Save it as `my-scheduler.yaml`:

[admin/sched/my-scheduler.yaml](#)



```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: my-scheduler
  namespace: kube-system
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: my-scheduler-as-kube-scheduler
subjects:
- kind: ServiceAccount
  name: my-scheduler
  namespace: kube-system
roleRef:
  kind: ClusterRole
  name: system:kube-scheduler
  apiGroup: rbac.authorization.k8s.io
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: my-scheduler-as-volume-scheduler
subjects:
- kind: ServiceAccount
  name: my-scheduler
  namespace: kube-system
roleRef:
  kind: ClusterRole
  name: system:volume-scheduler
  apiGroup: rbac.authorization.k8s.io
---
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    component: scheduler
```

```

    tier: control-plane
    name: my-scheduler
    namespace: kube-system
spec:
  selector:
    matchLabels:
      component: scheduler
      tier: control-plane
  replicas: 1
  template:
    metadata:
      labels:
        component: scheduler
        tier: control-plane
        version: second
    spec:
      serviceAccountName: my-scheduler
      containers:
        - command:
            - /usr/local/bin/kube-scheduler
            - --address=0.0.0.0
            - --leader-elect=false
            - --scheduler-name=my-scheduler
          image: gcr.io/my-gcp-project/my-kube-scheduler:1.0
          livenessProbe:
            httpGet:
              path: /healthz
              port: 10251
            initialDelaySeconds: 15
          name: kube-second-scheduler
          readinessProbe:
            httpGet:
              path: /healthz
              port: 10251
        resources:
          requests:
            cpu: '0.1'
        securityContext:
          privileged: false
        volumeMounts: []
      hostNetwork: false
      hostPID: false
      volumes: []

```

*An important thing to note here is that the name of the scheduler specified as an argument to the scheduler command in the container spec should be unique. This is the name that is matched against the value of the optional spec.schedulerName on pods, to determine whether this scheduler is responsible for scheduling a particular pod.*

Note also that we created a dedicated service account `my-scheduler` and bind the cluster role `system:kube-scheduler` to it so that it can acquire the same privileges as `kube-scheduler`.

Please see the [kube-scheduler documentation](#) for detailed description of other command line arguments.

## Run the second scheduler in the cluster

In order to run your scheduler in a Kubernetes cluster, just create the deployment specified in the config above in a Kubernetes cluster:

```
kubectl create -f my-scheduler.yaml
```

Verify that the scheduler pod is running:

```
kubectl get pods --namespace=kube-system
```

NAME	READY	
STATUS	RESTARTS	AGE
...		
<code>my-scheduler-lnf4s-4744f</code>		
<code>Running</code>	<code>0</code>	<code>2m</code>
...		

You should see a "Running" `my-scheduler` pod, in addition to the default `kube-scheduler` pod in this list.

## Enable leader election

To run multiple-scheduler with leader election enabled, you must do the following:

First, update the following fields in your YAML file:

- `--leader-elect=true`
- `--lock-object-namespace=<lock-object-namespace>`
- `--lock-object-name=<lock-object-name>`

**Note:** The control plane creates the lock objects for you, but the namespace must already exist. You can use the `kube-system` namespace.

If RBAC is enabled on your cluster, you must update the `system:kube-scheduler` cluster role. Add your scheduler name to the `resourceNames` of the rule applied for `endpoints` and `leases` resources, as in the following example:

```
kubectl edit clusterrole system:kube-scheduler
```

[admin/sched/clusterrole.yaml](#)



```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  annotations:
    rbac.authorization.kubernetes.io/autoupdate: "true"
  labels:
    kubernetes.io/bootstrapping: rbac-defaults
  name: system:kube-scheduler
rules:
- apiGroups:
  - coordination.k8s.io
  resources:
  - leases
  verbs:
  - create
- apiGroups:
  - coordination.k8s.io
  resourceNames:
  - kube-scheduler
  - my-scheduler
  resources:
  - leases
  verbs:
  - get
  - update
- apiGroups:
  - ""
  resourceNames:
  - kube-scheduler
  - my-scheduler
  resources:
  - endpoints
  verbs:
  - delete
  - get
  - patch
  - update
```

## **Specify schedulers for pods**

*Now that our second scheduler is running, let's create some pods, and direct them to be scheduled by either the default scheduler or the one we just deployed. In order to schedule a given pod using a specific scheduler, we specify the name of the scheduler in that pod spec. Let's look at three examples.*

- Pod spec without any scheduler name

[admin/sched/pod1.yaml](#)



```
apiVersion: v1
kind: Pod
metadata:
  name: no-annotation
  labels:
    name: multischeduler-example
spec:
  containers:
    - name: pod-with-no-annotation-container
      image: k8s.gcr.io/pause:2.0
```

*When no scheduler name is supplied, the pod is automatically scheduled using the default-scheduler.*

*Save this file as pod1.yaml and submit it to the Kubernetes cluster.*

```
kubectl create -f pod1.yaml
```

- Pod spec with default-scheduler

[admin/sched/pod2.yaml](#)



```
apiVersion: v1
kind: Pod
metadata:
  name: annotation-default-scheduler
  labels:
    name: multischeduler-example
spec:
  schedulerName: default-scheduler
  containers:
    - name: pod-with-default-annotation-container
      image: k8s.gcr.io/pause:2.0
```

A scheduler is specified by supplying the scheduler name as a value to `spec.schedulerName`. In this case, we supply the name of the default scheduler which is `default-scheduler`.

Save this file as `pod2.yaml` and submit it to the Kubernetes cluster.

```
kubectl create -f pod2.yaml
```

- Pod spec with `my-scheduler`

[admin/sched/pod3.yaml](#)



```
apiVersion: v1
kind: Pod
metadata:
  name: annotation-second-scheduler
  labels:
    name: multischeduler-example
spec:
  schedulerName: my-scheduler
  containers:
  - name: pod-with-second-annotation-container
    image: k8s.gcr.io/pause:2.0
```

In this case, we specify that this pod should be scheduled using the scheduler that we deployed - `my-scheduler`. Note that the value of `spec.schedulerName` should match the name supplied to the `scheduler` command as an argument in the deployment config for the scheduler.

Save this file as `pod3.yaml` and submit it to the Kubernetes cluster.

```
kubectl create -f pod3.yaml
```

Verify that all three pods are running.

```
kubectl get pods
```

## **Verifying that the pods were scheduled using the desired schedulers**

In order to make it easier to work through these examples, we did not verify that the pods were actually scheduled using the desired schedulers. We can verify that by changing the order of pod and deployment config submissions above. If we submit all the pod configs to a Kubernetes cluster before submitting the scheduler deployment config, we see that the pod `annotation-second-scheduler` remains in "Pending" state forever while the other two pods get scheduled. Once we submit the scheduler deployment config and our new scheduler

*starts running, the annotation-second-scheduler pod gets scheduled as well.*

*Alternatively, one could just look at the "Scheduled" entries in the event logs to verify that the pods were scheduled by the desired schedulers.*

```
kubectl get events
```

## 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](#).

Last modified August 07, 2020 at 8:40 PM PST: [Tune links in tasks section \(2/2\) \(92ae1a9cf\)](#)

[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Package the scheduler](#)
- [Define a Kubernetes Deployment for the scheduler](#)
- [Run the second scheduler in the cluster](#)
  - [Enable leader election](#)
  - [Specify schedulers for pods](#)
    - [Verifying that the pods were scheduled using the desired schedulers](#)

# Use an HTTP Proxy to Access the Kubernetes API

This page shows how to use an HTTP proxy to access the Kubernetes API.

## Before you begin

You need to have a Kubernetes cluster, and the kubectl command-line tool must be configured to communicate with your cluster. If you do not already have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:

- [Katacoda](#)
- [Play with Kubernetes](#)

*To check the version, enter `kubectl version`.*

*If you do not already have an application running in your cluster, start a Hello world application by entering this command:*

```
kubectl create deployment node-hello --image=gcr.io/google-samples/node-hello:1.0 --port=8080
```

## **Using `kubectl` to start a proxy server**

*This command starts a proxy to the Kubernetes API server:*

```
kubectl proxy --port=8080
```

## **Exploring the Kubernetes API**

*When the proxy server is running, you can explore the API using `curl`, `wget`, or a browser.*

*Get the API versions:*

```
curl http://localhost:8080/api/
```

*The output should look similar to this:*

```
{  
  "kind": "APIVersions",  
  "versions": [  
    "v1"  
  ],  
  "serverAddressByClientCIDRs": [  
    {  
      "clientCIDR": "0.0.0.0/0",  
      "serverAddress": "10.0.2.15:8443"  
    }  
  ]  
}
```

*Get a list of pods:*

```
curl http://localhost:8080/api/v1/namespaces/default/pods
```

The output should look similar to this:

```
{  
  "kind": "PodList",  
  "apiVersion": "v1",  
  "metadata": {  
    "resourceVersion": "33074"  
  },  
  "items": [  
    {  
      "metadata": {  
        "name": "kubernetes-bootcamp-2321272333-ix8pt",  
        "generateName": "kubernetes-bootcamp-2321272333-",  
        "namespace": "default",  
        "uid": "ba21457c-6b1d-11e6-85f7-1ef9f1dab92b",  
        "resourceVersion": "33003",  
        "creationTimestamp": "2016-08-25T23:43:30Z",  
        "labels": {  
          "pod-template-hash": "2321272333",  
          "run": "kubernetes-bootcamp"  
        },  
        ...  
      }  
    ]  
  ]  
}
```

## What's next

Learn more about [kubectl 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](#).

Last modified July 09, 2020 at 10:21 AM PST: [Avoid using deprecated commands in task \(621e5e723\)](#)

[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Using kubectl to start a proxy server](#)
- [Exploring the Kubernetes API](#)
- [What's next](#)

# Set up Konnectivity service

The Konnectivity service provides a TCP level proxy for the control plane to cluster communication.

## Before you begin

You need to have a Kubernetes cluster, and the `kubectl` command-line tool must be configured to communicate with your cluster. If you do not already have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:

- [Katacoda](#)
- [Play with Kubernetes](#)

## Configure the Konnectivity service

The following steps require an egress configuration, for example:

[admin/konnectivity/egress-selector-configuration.yaml](#)  


```
apiVersion: apiserver.k8s.io/v1beta1
kind: EgressSelectorConfiguration
egressSelections:
# Since we want to control the egress traffic to the
# cluster, we use the
# "cluster" as the name. Other supported values are "etcd",
# and "master".
- name: cluster
  connection:
    # This controls the protocol between the API Server and
    # the Konnectivity
    # server. Supported values are "GRPC" and "HTTPConnect".
    There is no
    # end user visible difference between the two modes. You
    need to set the
    # Konnectivity server to work in the same mode.
  proxyProtocol: GRPC
  transport:
    # This controls what transport the API Server uses to
    # communicate with the
    # Konnectivity server. UDS is recommended if the
    # Konnectivity server
    # locates on the same machine as the API Server. You
    # need to configure the
    # Konnectivity server to listen on the same UDS socket.
    # The other supported transport is "tcp". You will
    # need to set up TLS
```

```

# config to secure the TCP transport.
uds:
  udsName: /etc/kubernetes/konnectivity-server/
  konnectivity-server.socket

```

You need to configure the API Server to use the Konnectivity service and direct the network traffic to the cluster nodes:

1. Make sure that the `ServiceAccountTokenVolumeProjection feature gate` is enabled. You can enable [service account token volume protection](#) by providing the following flags to the `kube-apiserver`:

```

--service-account-issuer=api
--service-account-signing-key-file=/etc/kubernetes/pki/
sa.key
--api-audiences=system:konnectivity-server

```

2. Create an egress configuration file such as `admin/konnectivity/egress-selector-configuration.yaml`.
3. Set the `--egress-selector-config-file` flag of the API Server to the path of your API Server egress configuration file.

Generate or obtain a certificate and `kubeconfig` for `konnectivity-server`. For example, you can use the OpenSSL command line tool to issue a X.509 certificate, using the cluster CA certificate `/etc/kubernetes/pki/ca.crt` from a control-plane host.

```

openssl req -subj "/CN=system:konnectivity-server" -new -
newkey rsa:2048 -nodes -out konnectivity.csr -keyout
konnectivity.key -out konnectivity.csr
openssl x509 -req -in konnectivity.csr -CA /etc/kubernetes/
pki/ca.crt -CAkey /etc/kubernetes/pki/ca.key -CAcreateserial
-out konnectivity.crt -days 375 -sha256
$ SERVER=$(kubectl config view -o jsonpath='{.clusters..server}')
)
kubectl --kubeconfig /etc/kubernetes/konnectivity-
server.conf config set-credentials system:konnectivity-
server --client-certificate konnectivity.crt --client-key
konnectivity.key --embed-certs=true
kubectl --kubeconfig /etc/kubernetes/konnectivity-
server.conf config set-cluster kubernetes --server "$SERVER"
--certificate-authority /etc/kubernetes/pki/ca.crt --embed-
certs=true
kubectl --kubeconfig /etc/kubernetes/konnectivity-
server.conf config set-context system:konnectivity-
server@kubernetes --cluster kubernetes --user
system:konnectivity-server
kubectl --kubeconfig /etc/kubernetes/konnectivity-
server.conf config use-context system:konnectivity-
server@kubernetes
rm -f konnectivity.crt konnectivity.key konnectivity.csr

```

Next, you need to deploy the Konnectivity server and agents. [kubernetes-sigs/apiserver-network-proxy](#) is a reference implementation.

Deploy the Konnectivity server on your control plane node. The provided `konnectivity-server.yaml` manifest assumes that the Kubernetes components are deployed as a [static Pod](#) in your cluster. If not, you can deploy the Konnectivity server as a DaemonSet.

[admin/konnectivity/konnectivity-server.yaml](#)



```
apiVersion: v1
kind: Pod
metadata:
  name: konnectivity-server
  namespace: kube-system
spec:
  priorityClassName: system-cluster-critical
  hostNetwork: true
  containers:
    - name: konnectivity-server-container
      image: us.gcr.io/k8s-artifacts-prod/kas-network-proxy/
proxy-server:v0.0.12
      command: ["/proxy-server"]
      args: [
        "--logtostderr=true",
        # This needs to be consistent with the value set
        in egressSelectorConfiguration.
        "--uds-name=/etc/kubernetes/konnectivity-server/
konnectivity-server.socket",
        # The following two lines assume the
        Konnectivity server is
        # deployed on the same machine as the apiserver,
        and the certs and
        # key of the API Server are at the specified
        location.
        "--cluster-cert=/etc/kubernetes/pki/
apiserver.crt",
        "--cluster-key=/etc/kubernetes/pki/
apiserver.key",
        # This needs to be consistent with the value set
        in egressSelectorConfiguration.
        "--mode=grpc",
        "--server-port=0",
        "--agent-port=8132",
        "--admin-port=8133",
        "--health-port=8134",
        "--agent-namespace=kube-system",
        "--agent-service-account=konnectivity-agent",
        "--kubeconfig=/etc/kubernetes/konnectivity-
server.conf",
        "--authentication-audience=system:konnectivity-
```

```

server"
    ]
  livenessProbe:
    httpGet:
      scheme: HTTP
      host: 127.0.0.1
      port: 8134
      path: /healthz
    initialDelaySeconds: 30
    timeoutSeconds: 60
  ports:
    - name: agentport
      containerPort: 8132
      hostPort: 8132
    - name: adminport
      containerPort: 8133
      hostPort: 8133
    - name: healthport
      containerPort: 8134
      hostPort: 8134
  volumeMounts:
    - name: k8s-certs
      mountPath: /etc/kubernetes/pki
      readOnly: true
    - name: kubeconfig
      mountPath: /etc/kubernetes/konnectivity-server.conf
      readOnly: true
    - name: konnectivity-uds
      mountPath: /etc/kubernetes/konnectivity-server
      readOnly: false
  volumes:
    - name: k8s-certs
      hostPath:
        path: /etc/kubernetes/pki
    - name: kubeconfig
      hostPath:
        path: /etc/kubernetes/konnectivity-server.conf
        type: FileOrCreate
    - name: konnectivity-uds
      hostPath:
        path: /etc/kubernetes/konnectivity-server
        type: DirectoryOrCreate

```

*Then deploy the Konnectivity agents in your cluster:*

[admin/konnectivity/konnectivity-agent.yaml](#)

```

apiVersion: apps/v1
# Alternatively, you can deploy the agents as Deployments.
# It is not necessary
# to have an agent on each node.

```

```

kind: DaemonSet
metadata:
  labels:
    addonmanager.kubernetes.io/mode: Reconcile
    k8s-app: konnectivity-agent
  namespace: kube-system
  name: konnectivity-agent
spec:
  selector:
    matchLabels:
      k8s-app: konnectivity-agent
  template:
    metadata:
      labels:
        k8s-app: konnectivity-agent
    spec:
      priorityClassName: system-cluster-critical
      tolerations:
        - key: "CriticalAddonsOnly"
          operator: "Exists"
      containers:
        - image: us.gcr.io/k8s-artifacts-prod/kas-network-
proxy/proxy-agent:v0.0.12
          name: konnectivity-agent
          command: ["/proxy-agent"]
          args: [
            "--logtostderr=true",
            "--ca-cert=/var/run/secrets/kubernetes.io/
serviceaccount/ca.crt",
            # Since the konnectivity server runs with
hostNetwork=true,
            # this is the IP address of the master
machine.
            "--proxy-server-host=35.225.206.7",
            "--proxy-server-port=8132",
            "--admin-server-port=8133",
            "--health-server-port=8134",
            "--service-account-token-path=/var/run/
secrets/tokens/konnectivity-agent-token"
          ]
      volumeMounts:
        - mountPath: /var/run/secrets/tokens
          name: konnectivity-agent-token
      livenessProbe:
        httpGet:
          port: 8134
          path: /healthz
        initialDelaySeconds: 15
        timeoutSeconds: 15
      serviceAccountName: konnectivity-agent
      volumes:
        - name: konnectivity-agent-token

```

```
projected:  
  sources:  
    - serviceAccountToken:  
        path: konnectivity-agent-token  
        audience: system:konnectivity-server
```

Last, if RBAC is enabled in your cluster, create the relevant RBAC rules:

[admin/konnectivity/konnectivity-rbac.yaml](#)



```
apiVersion: rbac.authorization.k8s.io/v1  
kind: ClusterRoleBinding  
metadata:  
  name: system:konnectivity-server  
  labels:  
    kubernetes.io/cluster-service: "true"  
    addonmanager.kubernetes.io/mode: Reconcile  
roleRef:  
  apiGroup: rbac.authorization.k8s.io  
  kind: ClusterRole  
  name: system:auth-delegator  
subjects:  
  - apiGroup: rbac.authorization.k8s.io  
    kind: User  
    name: system:konnectivity-server  
---  
apiVersion: v1  
kind: ServiceAccount  
metadata:  
  name: konnectivity-agent  
  namespace: kube-system  
  labels:  
    kubernetes.io/cluster-service: "true"  
    addonmanager.kubernetes.io/mode: Reconcile
```

## 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](#).

Last modified November 13, 2020 at 8:52 AM PST: [Add missing steps to configure konnectivity-server \(#24141\) \(798b5c9f2\)](#)

[Edit this page](#) [Create child page](#) [Create an issue](#)

◦ [Before you begin](#)

- [Configure the Konnectivity service](#)

# TLS

Understand how to protect traffic within your cluster using Transport Layer Security (TLS).

---

[Configure Certificate Rotation for the Kubelet](#)

[Manage TLS Certificates in a Cluster](#)

[Manual Rotation of CA Certificates](#)

## Configure Certificate Rotation for the Kubelet

This page shows how to enable and configure certificate rotation for the kubelet.

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

### Before you begin

- Kubernetes version 1.8.0 or later is required

### Overview

The kubelet uses certificates for authenticating to the Kubernetes API. By default, these certificates are issued with one year expiration so that they do not need to be renewed too frequently.

Kubernetes 1.8 contains [kubelet certificate rotation](#), a beta feature that will automatically generate a new key and request a new certificate from the Kubernetes API as the current certificate approaches expiration. Once the new certificate is available, it will be used for authenticating connections to the Kubernetes API.

### Enabling client certificate rotation

The kubelet process accepts an argument `--rotate-certificates` that controls if the kubelet will automatically request a new certificate as the expiration of the certificate currently in use approaches.

The `kube-controller-manager` process accepts an argument `--cluster-signing-duration` (`--experimental-cluster-signing-`

*duration prior to 1.19) that controls how long certificates will be issued for.*

## **Understanding the certificate rotation configuration**

*When a kubelet starts up, if it is configured to bootstrap (using the --bootstrap-kubeconfig flag), it will use its initial certificate to connect to the Kubernetes API and issue a certificate signing request. You can view the status of certificate signing requests using:*

```
kubectl get csr
```

*Initially a certificate signing request from the kubelet on a node will have a status of Pending. If the certificate signing requests meets specific criteria, it will be auto approved by the controller manager, then it will have a status of Approved. Next, the controller manager will sign a certificate, issued for the duration specified by the --cluster-signing-duration parameter, and the signed certificate will be attached to the certificate signing requests.*

*The kubelet will retrieve the signed certificate from the Kubernetes API and write that to disk, in the location specified by --cert-dir. Then the kubelet will use the new certificate to connect to the Kubernetes API.*

*As the expiration of the signed certificate approaches, the kubelet will automatically issue a new certificate signing request, using the Kubernetes API. Again, the controller manager will automatically approve the certificate request and attach a signed certificate to the certificate signing request. The kubelet will retrieve the new signed certificate from the Kubernetes API and write that to disk. Then it will update the connections it has to the Kubernetes API to reconnect using the new certificate.*

## **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](#).

Last modified May 21, 2020 at 11:52 AM PST: [Rotate kubelet client certificates GA \(bc2ed912d\)](#)

[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Overview](#)
- [Enabling client certificate rotation](#)

- [Understanding the certificate rotation configuration](#)

# Manage TLS Certificates in a Cluster

Kubernetes provides a `certificates.k8s.io` API, which lets you provision TLS certificates signed by a Certificate Authority (CA) that you control. These CA and certificates can be used by your workloads to establish trust.

`certificates.k8s.io` API uses a protocol that is similar to the [ACME draft](#).

**Note:** Certificates created using the `certificates.k8s.io` API are signed by a dedicated CA. It is possible to configure your cluster to use the cluster root CA for this purpose, but you should never rely on this. Do not assume that these certificates will validate against the cluster root CA.

## Before you begin

You need to have a Kubernetes cluster, and the `kubectl` command-line tool must be configured to communicate with your cluster. If you do not already have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:

- [Katacoda](#)
- [Play with Kubernetes](#)

To check the version, enter `kubectl version`.

## Trusting TLS in a Cluster

Trusting the custom CA from an application running as a pod usually requires some extra application configuration. You will need to add the CA certificate bundle to the list of CA certificates that the TLS client or server trusts. For example, you would do this with a golang TLS config by parsing the certificate chain and adding the parsed certificates to the `RootCAs` field in the [tls.Config](#) struct.

You can distribute the CA certificate as a [ConfigMap](#) that your pods have access to use.

## Requesting a Certificate

The following section demonstrates how to create a TLS certificate for a Kubernetes service accessed through DNS.

**Note:** This tutorial uses CFSSL: Cloudflare's PKI and TLS toolkit [click here](#) to know more.

## Download and install CFSSL

The cfssl tools used in this example can be downloaded at <https://pkg.cfssl.org/>.

## Create a Certificate Signing Request

Generate a private key and certificate signing request (or CSR) by running the following command:

```
cat <<EOF | cfssl genkey - | cfssljson -bare server
{
  "hosts": [
    "my-svc.my-namespace.svc.cluster.local",
    "my-pod.my-namespace.pod.cluster.local",
    "192.0.2.24",
    "10.0.34.2"
  ],
  "CN": "system:node:my-pod.my-namespace.pod.cluster.local",
  "key": {
    "algo": "ecdsa",
    "size": 256
  },
  "names": [
    {
      "0": "system:nodes"
    }
  ]
}
EOF
```

Where 192.0.2.24 is the service's cluster IP, my-svc.my-namespace.svc.cluster.local is the service's DNS name, 10.0.34.2 is the pod's IP and my-pod.my-namespace.pod.cluster.local is the pod's DNS name. You should see the following output:

```
2017/03/21 06:48:17 [INFO] generate received request
2017/03/21 06:48:17 [INFO] received CSR
```

```
2017/03/21 06:48:17 [INFO] generating key: ecdsa-256
2017/03/21 06:48:17 [INFO] encoded CSR
```

This command generates two files; it generates `server.csr` containing the PEM encoded [pkcs#10](#) certification request, and `server-key.pem` containing the PEM encoded key to the certificate that is still to be created.

## Create a Certificate Signing Request object to send to the Kubernetes API

Generate a CSR yaml blob and send it to the apiserver by running the following command:

```
cat <<EOF | kubectl apply -f -
apiVersion: certificates.k8s.io/v1
kind: CertificateSigningRequest
metadata:
  name: my-svc.my-namespace
spec:
  request: $(cat server.csr | base64 | tr -d '\n')
  signerName: kubernetes.io/kubelet-serving
  usages:
  - digital signature
  - key encipherment
  - server auth
EOF
```

Notice that the `server.csr` file created in step 1 is base64 encoded and stashed in the `.spec.request` field. We are also requesting a certificate with the "digital signature", "key encipherment", and "server auth" key usages, signed by the `kubernetes.io/kubelet-serving` signer. A specific `signerName` must be requested. View documentation for [supported signer names](#) for more information.

The CSR should now be visible from the API in a Pending state. You can see it by running:

```
kubectl describe csr my-svc.my-namespace
```

Name:	<code>my-svc.my-namespace</code>
Labels:	<code>&lt;none&gt;</code>
Annotations:	<code>&lt;none&gt;</code>
CreationTimestamp:	<code>Tue, 21 Mar 2017 07:03:51 -0700</code>
Requesting User:	<code>yourname@example.com</code>

```
Status:          Pending
Subject:
  Common Name:  my-svc.my-namespace.svc.cluster.local
  Serial Number:
Subject Alternative Names:
  DNS Names:   my-svc.my-namespace.svc.cluster.local
  IP Addresses: 192.0.2.24
                           10.0.34.2
Events: <none>
```

## **Get the Certificate Signing Request Approved**

*Approving the certificate signing request is either done by an automated approval process or on a one off basis by a cluster administrator. More information on what this involves is covered below.*

## **Download the Certificate and Use It**

*Once the CSR is signed and approved you should see the following:*

```
kubectl get csr
```

NAME	AGE	REQUESTOR
<i>my-svc.my-namespace</i>	<i>10m</i>	<i>yourname@example.com</i>
<i>Approved, Issued</i>		

*You can download the issued certificate and save it to a `server.crt` file by running the following:*

```
kubectl get csr my-svc.my-namespace -o jsonpath='{.status.certificate}' \
| base64 --decode > server.crt
```

*Now you can use `server.crt` and `server-key.pem` as the keypair to start your HTTPS server.*

## **Approving Certificate Signing Requests**

A Kubernetes administrator (with appropriate permissions) can manually approve (or deny) Certificate Signing Requests by using the `kubectl certificate approve` and `kubectl certificate deny` commands. However if you intend to make heavy usage of this API, you might consider writing an automated certificates controller.

Whether a machine or a human using `kubectl` as above, the role of the approver is to verify that the CSR satisfies two requirements:

1. *The subject of the CSR controls the private key used to sign the CSR. This addresses the threat of a third party masquerading as an authorized subject. In the above example, this step would be to verify that the pod controls the private key used to generate the CSR.*
2. *The subject of the CSR is authorized to act in the requested context. This addresses the threat of an undesired subject joining the cluster. In the above example, this step would be to verify that the pod is allowed to participate in the requested service.*

*If and only if these two requirements are met, the approver should approve the CSR and otherwise should deny the CSR.*

## **A Word of Warning on the Approval Permission**

*The ability to approve CSRs decides who trusts whom within your environment. The ability to approve CSRs should not be granted broadly or lightly. The requirements of the challenge noted in the previous section and the repercussions of issuing a specific certificate should be fully understood before granting this permission.*

## **A Note to Cluster Administrators**

*This tutorial assumes that a signer is setup to serve the certificates API. The Kubernetes controller manager provides a default implementation of a signer. To enable it, pass the `--cluster-signing-cert-file` and `--cluster-signing-key-file` parameters to the controller manager with paths to your Certificate Authority's keypair.*

## **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](#).

Last modified October 07, 2020 at 11:17 PM PST: [Update managing-tls-in-a-cluster.md to include fields in certificate signing request \(or CSR\) required by kubernetes.io/kubelet-serving signer policy \(ea3ed2b6e\)](#)  
[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Trusting TLS in a Cluster](#)
- [Requesting a Certificate](#)
- [Download and install CFSSL](#)
- [Create a Certificate Signing Request](#)
- [Create a Certificate Signing Request object to send to the Kubernetes API](#)
- [Get the Certificate Signing Request Approved](#)
- [Download the Certificate and Use It](#)
- [Approving Certificate Signing Requests](#)
- [A Word of Warning on the Approval Permission](#)
- [A Note to Cluster Administrators](#)

# **Manual Rotation of CA Certificates**

This page shows how to manually rotate the certificate authority (CA) certificates.

## **Before you begin**

You need to have a Kubernetes cluster, and the `kubectl` command-line tool must be configured to communicate with your cluster. If you do not already have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:

- [Katacoda](#)
- [Play with Kubernetes](#)

Your Kubernetes server must be at or later than version v1.13. To check the version, enter `kubectl version`.

- For more information about authentication in Kubernetes, see [Authenticating](#).
- For more information about best practices for CA certificates, see [Single root CA](#).

## **Rotate the CA certificates manually**

**Caution:**

*Make sure to back up your certificate directory along with configuration files and any other necessary files.*

*This approach assumes operation of the Kubernetes control plane in a HA configuration with multiple API servers.*

*Graceful termination of the API server is also assumed so clients can cleanly disconnect from one API server and reconnect to another.*

*Configurations with a single API server will experience unavailability while the API server is being restarted.*

1. *Distribute the new CA certificates and private keys (ex: ca.crt, ca.key, front-proxy-ca.crt, and front-proxy-ca.key) to all your control plane nodes in the Kubernetes certificates directory.*
2. *Update [kube-controller-manager](#)'s --root-ca-file to include both old and new CA. Then restart the component.*

*Any service account created after this point will get secrets that include both old and new CAs.*

**Note:** The files specified by the kube-controller-manager flags --client-ca-file and --cluster-signing-cert-file cannot be CA bundles. If these flags and --root-ca-file point to the same ca.crt file which is now a bundle (includes both old and new CA) you will face an error. To workaround this problem you can copy the new CA to a separate file and make the flags --client-ca-file and --cluster-signing-cert-file point to the copy. Once ca.crt is no longer a bundle you can restore the problem flags to point to ca.crt and delete the copy.

3. *Update all service account tokens to include both old and new CA certificates.*

*If any pods are started before new CA is used by API servers, they will get this update and trust both old and new CAs.*

```
base64_encoded_ca=$(base64 <path to file containing both old and new CAs>"

for namespace in $(kubectl get ns --no-headers | awk '{print $1}'); do
    for token in $(kubectl get secrets --namespace "$namespace" --field-selector type=kubernetes.io/service-account-token -o name); do
        kubectl get $token --namespace "$namespace" -o yaml | \
            /bin/sed "s/\\(ca.crt:\\).*/\\1 ${base64_encoded_ca}" | \
            kubectl apply -f -
```

*done*

*done*

4. *Restart all pods using in-cluster configs (ex: kube-proxy, coredns, etc) so they can use the updated certificate authority data from ServiceAccount secrets.*
  - *Make sure coredns, kube-proxy and other pods using in-cluster configs are working as expected.*
5. *Append the both old and new CA to the file against --client-ca-file and --kubelet-certificate-authority flag in the kube-apiserver configuration.*
6. *Append the both old and new CA to the file against --client-ca-file flag in the kube-scheduler configuration.*
7. *Update certificates for user accounts by replacing the content of client-certificate-data and client-key-data respectively.*

*For information about creating certificates for individual user accounts, see [Configure certificates for user accounts](#).*

*Additionally, update the certificate-authority-data section in the kubeconfig files, respectively with Base64-encoded old and new certificate authority data*

8. *Follow below steps in a rolling fashion.*

1. *Restart any other aggregated api servers or webhook handlers to trust the new CA certificates.*
2. *Restart the kubelet by update the file against clientCAFile in kubelet configuration and certificate-authority-data in kubelet.conf to use both the old and new CA on all nodes.*

*If your kubelet is not using client certificate rotation update client-certificate-data and client-key-data in kubelet.conf on all nodes along with the kubelet client certificate file usually found in /var/lib/kubelet/pki.*

3. *Restart API servers with the certificates (apiserver.crt, api-server-kubelet-client.crt and front-proxy-client.crt) signed by new CA. You can use the existing private keys or new private keys. If you changed the private keys then update these in the Kubernetes certificates directory as well.*

*Since the pod trusts both old and new CAs, there will be a momentarily disconnection after which the pod's kube client will reconnect to the new API server that uses the certificate signed by the new CA.*

- *Restart Scheduler to use the new CAs.*

*Make sure control plane components logs no TLS errors.*

**Note:** To generate certificates and private keys for your cluster using the `openssl` command line tool, see [Certificates \(openssl\)](#). You can also use [cfssl](#).

4. Annotate any Daemonsets and Deployments to trigger pod replacement in a safer rolling fashion.

*Example:*

```
for namespace in $(kubectl get namespace -o jsonpath='{.items[*].metadata.name}'); do
    for name in $(kubectl get deployments -n $namespace -o jsonpath='{.items[*].metadata.name}'); do
        kubectl patch deployment -n ${namespace} ${name} -p '{"spec":{"template":{"metadata":{"annotations":{"ca-rotation": "1"}}}}';
    done
    for name in $(kubectl get daemonset -n $namespace -o jsonpath='{.items[*].metadata.name}'); do
        kubectl patch daemonset -n ${namespace} ${name} -p '{"spec":{"template":{"metadata":{"annotations":{"ca-rotation": "1"}}}}';
    done
done
```

**Note:** To limit the number of concurrent disruptions that your application experiences, see [configure pod disruption budget](#).

9. If your cluster is using bootstrap tokens to join nodes, update the `ConfigMap cluster-info` in the `kube-public` namespace with new CA.

```
base64_encoded_ca=$(base64 /etc/kubernetes/pki/ca.crt)

kubectl get cm/cluster-info --namespace kube-public -o yaml | \
/bin/sed "s/^(certificate-authority-data:).*$/\1 ${base64_encoded_ca}" | \
kubectl apply -f -
```

10. Verify the cluster functionality.

1. Validate the logs from control plane components, along with the `kubelet` and the `kube-proxy` are not throwing any `tls` errors, see [looking at the logs](#).
2. Validate logs from any aggregated api servers and pods using in-cluster config.

- Once the cluster functionality is successfully verified:
11.
    1. Update all service account tokens to include new CA certificate only.
      - All pods using an in-cluster kubeconfig will eventually need to be restarted to pick up the new SA secret for the old CA to be completely untrusted.
    2. Restart the control plane components by removing the old CA from the kubeconfig files and the files against `--client-ca-file`, `--root-ca-file` flags resp.
    3. Restart kubelet by removing the old CA from file against the `clientCAFile` flag and kubelet kubeconfig file.

## 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](#).

Last modified November 16, 2020 at 11:04 PM PST: [manual-rotation-of-ca-certificates: use kube-controller-manager naming \(c6ac78300\)](#)

[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Rotate the CA certificates manually](#)

## Manage Cluster Daemons

Perform common tasks for managing a DaemonSet, such as performing a rolling update.

---

[Perform a Rolling Update on a DaemonSet](#)

[Perform a Rollback on a DaemonSet](#)

## Perform a Rolling Update on a DaemonSet

This page shows how to perform a rolling update on a DaemonSet.

# **Before you begin**

- The DaemonSet rolling update feature is only supported in Kubernetes version 1.6 or later.

## **DaemonSet Update Strategy**

DaemonSet has two update strategy types:

- *OnDelete*: With *OnDelete* update strategy, after you update a DaemonSet template, new DaemonSet pods will only be created when you manually delete old DaemonSet pods. This is the same behavior of DaemonSet in Kubernetes version 1.5 or before.
- *RollingUpdate*: This is the default update strategy. With *RollingUpdate* update strategy, after you update a DaemonSet template, old DaemonSet pods will be killed, and new DaemonSet pods will be created automatically, in a controlled fashion. At most one pod of the DaemonSet will be running on each node during the whole update process.

## **Performing a Rolling Update**

To enable the rolling update feature of a DaemonSet, you must set its `.spec.updateStrategy.type` to `RollingUpdate`.

You may want to set `.spec.updateStrategy.rollingUpdate.maxUnavailable` (default to 1) and `.spec.minReadySeconds` (default to 0) as well.

### **Creating a DaemonSet with RollingUpdate update strategy**

This YAML file specifies a DaemonSet with an update strategy as 'RollingUpdate'

[controllers/fluentd-daemonset.yaml](#)  


```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: fluentd-elasticsearch
  namespace: kube-system
  labels:
    k8s-app: fluentd-logging
spec:
  selector:
    matchLabels:
      name: fluentd-elasticsearch
  updateStrategy:
```

```

type: RollingUpdate
rollingUpdate:
  maxUnavailable: 1
template:
  metadata:
    labels:
      name: fluentd-elasticsearch
spec:
  tolerations:
    # this toleration is to have the daemonset runnable on
    master nodes
    # remove it if your masters can't run pods
    - key: node-role.kubernetes.io/master
      effect: NoSchedule
  containers:
    - name: fluentd-elasticsearch
      image: quay.io/fluentd_elasticsearch/fluentd:v2.5.2
      volumeMounts:
        - name: varlog
          mountPath: /var/log
        - name: varlibdockercontainers
          mountPath: /var/lib/docker/containers
          readOnly: true
  terminationGracePeriodSeconds: 30
volumes:
  - name: varlog
    hostPath:
      path: /var/log
  - name: varlibdockercontainers
    hostPath:
      path: /var/lib/docker/containers

```

After verifying the update strategy of the DaemonSet manifest, create the DaemonSet:

```
kubectl create -f https://k8s.io/examples/controllers/
fluentd-daemonset.yaml
```

Alternatively, use `kubectl apply` to create the same DaemonSet if you plan to update the DaemonSet with `kubectl apply`.

```
kubectl apply -f https://k8s.io/examples/controllers/fluentd-
daemonset.yaml
```

## **Checking DaemonSet RollingUpdate update strategy**

Check the update strategy of your DaemonSet, and make sure it's set to `RollingUpdate`:

```
kubectl get ds/fluentd-elasticsearch -o go-template='{{.spec.
updateStrategy.type}}{{"\n"}}' -n kube-system
```

If you haven't created the DaemonSet in the system, check your DaemonSet manifest with the following command instead:

```
kubectl apply -f https://k8s.io/examples/controllers/fluentd-daemonset.yaml --dry-run=client -o go-template='{{.spec.updateStrategy.type}}{{"\n"}}'
```

The output from both commands should be:

*RollingUpdate*

If the output isn't RollingUpdate, go back and modify the DaemonSet object or manifest accordingly.

## Updating a DaemonSet template

Any updates to a RollingUpdate DaemonSet .spec.template will trigger a rolling update. Let's update the DaemonSet by applying a new YAML file. This can be done with several different kubectl commands.

[controllers/fluentd-daemonset-update.yaml](#)  


```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: fluentd-elasticsearch
  namespace: kube-system
  labels:
    k8s-app: fluentd-logging
spec:
  selector:
    matchLabels:
      name: fluentd-elasticsearch
  updateStrategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 1
  template:
    metadata:
      labels:
        name: fluentd-elasticsearch
  spec:
    tolerations:
      # this toleration is to have the daemonset runnable on master nodes
      # remove it if your masters can't run pods
      - key: node-role.kubernetes.io/master
        effect: NoSchedule
    containers:
      - name: fluentd-elasticsearch
        image: quay.io/fluentd_elasticsearch/fluentd:v2.5.2
```

```
resources:
  limits:
    memory: 200Mi
  requests:
    cpu: 100m
    memory: 200Mi
volumeMounts:
- name: varlog
  mountPath: /var/log
- name: varlibdockercontainers
  mountPath: /var/lib/docker/containers
  readOnly: true
terminationGracePeriodSeconds: 30
volumes:
- name: varlog
  hostPath:
    path: /var/log
- name: varlibdockercontainers
  hostPath:
    path: /var/lib/docker/containers
```

## **Declarative commands**

If you update DaemonSets using [configuration files](#), use `kubectl apply`:

```
kubectl apply -f https://k8s.io/examples/controllers/fluentd-
daemonset-update.yaml
```

## **Imperative commands**

If you update DaemonSets using [imperative commands](#), use `kubectl edit`:

```
kubectl edit ds/fluentd-elasticsearch -n kube-system
```

### **Updating only the container image**

If you just need to update the container image in the DaemonSet template, i.e. `.spec.template.spec.containers[*].image`, use `kubectl set image`:

```
kubectl set image ds/fluentd-elasticsearch fluentd-
elasticsearch=quay.io/fluentd_elasticsearch/fluentd:v2.6.0 -
n kube-system
```

## **Watching the rolling update status**

Finally, watch the rollout status of the latest DaemonSet rolling update:

```
kubectl rollout status ds/fluentd-elasticsearch -n kube-system
```

When the rollout is complete, the output is similar to this:

```
daemonset "fluentd-elasticsearch" successfully rolled out
```

## Troubleshooting

### DaemonSet rolling update is stuck

Sometimes, a DaemonSet rolling update may be stuck. Here are some possible causes:

#### Some nodes run out of resources

The rollout is stuck because new DaemonSet pods can't be scheduled on at least one node. This is possible when the node is [running out of resources](#).

When this happens, find the nodes that don't have the DaemonSet pods scheduled on by comparing the output of `kubectl get nodes` and the output of:

```
kubectl get pods -l name=fluentd-elasticsearch -o wide -n kube-system
```

Once you've found those nodes, delete some non-DaemonSet pods from the node to make room for new DaemonSet pods.

**Note:** This will cause service disruption when deleted pods are not controlled by any controllers or pods are not replicated. This does not respect [PodDisruptionBudget](#) either.

#### Broken rollout

If the recent DaemonSet template update is broken, for example, the container is crash looping, or the container image doesn't exist (often due to a typo), DaemonSet rollout won't progress.

To fix this, just update the DaemonSet template again. New rollout won't be blocked by previous unhealthy rollouts.

#### Clock skew

If `.spec.minReadySeconds` is specified in the DaemonSet, clock skew between master and nodes will make DaemonSet unable to detect the right rollout progress.

## **Clean up**

*Delete DaemonSet from a namespace :*

```
kubectl delete ds fluentd-elasticsearch -n kube-system
```

## **What's next**

- See [Task: Performing a rollback on a DaemonSet](#)
- See [Concepts: Creating a DaemonSet to adopt existing DaemonSet pods](#)

## **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](#).*

*Last modified August 07, 2020 at 8:40 PM PST: [Tune links in tasks section \(2/2\) \(92ae1a9cf\)](#)*

[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [DaemonSet Update Strategy](#)
- [Performing a Rolling Update](#)
  - [Creating a DaemonSet with RollingUpdate update strategy](#)
  - [Checking DaemonSet RollingUpdate update strategy](#)
  - [Updating a DaemonSet template](#)
  - [Watching the rolling update status](#)
- [Troubleshooting](#)
  - [DaemonSet rolling update is stuck](#)
- [Clean up](#)
- [What's next](#)

## **Perform a Rollback on a DaemonSet**

*This page shows how to perform a rollback on a [DaemonSet](#).*

### **Before you begin**

*You need to have a Kubernetes cluster, and the kubectl command-line tool must be configured to communicate with your cluster. If you do not*

already have a cluster, you can create one by using [minikube](#) or you can use one of these Kubernetes playgrounds:

- [Katacoda](#)
- [Play with Kubernetes](#)

Your Kubernetes server must be at or later than version 1.7. To check the version, enter `kubectl version`.

You should already know how to [perform a rolling update on a DaemonSet](#).

## **Performing a rollback on a DaemonSet**

### **Step 1: Find the DaemonSet revision you want to roll back to**

You can skip this step if you just want to roll back to the last revision.

List all revisions of a DaemonSet:

```
kubectl rollout history daemonset <daemonset-name>
```

This returns a list of DaemonSet revisions:

```
daemonsets "<daemonset-name>"  
REVISION      CHANGE-CAUSE  
1             ...  
2             ...  
...           ...
```

- Change cause is copied from DaemonSet annotation `kubernetes.io/change-cause` to its revisions upon creation. You may specify `--record=true` in `kubectl` to record the command executed in the change cause annotation.

To see the details of a specific revision:

```
kubectl rollout history daemonset <daemonset-name> --  
revision=1
```

This returns the details of that revision:

```
daemonsets "<daemonset-name>" with revision #1  
Pod Template:
```

```
Labels:      foo=bar
Containers:
app:
  Image:      ...
  Port:       ...
  Environment: ...
  Mounts:     ...
  Volumes:    ...
```

## **Step 2: Roll back to a specific revision**

```
# Specify the revision number you get from Step 1 in --to-revision
kubectl rollout undo daemonset <daemonset-name> --to-revision=<revision>
```

If it succeeds, the command returns:

```
daemonset "<daemonset-name>" rolled back
```

**Note:** If `--to-revision` flag is not specified, `kubectl` picks the most recent revision.

## **Step 3: Watch the progress of the DaemonSet rollback**

`kubectl rollout undo daemonset` tells the server to start rolling back the DaemonSet. The real rollback is done asynchronously inside the cluster [control plane](#).

To watch the progress of the rollback:

```
kubectl rollout status ds/<daemonset-name>
```

When the rollback is complete, the output is similar to:

```
daemonset "<daemonset-name>" successfully rolled out
```

## **Understanding DaemonSet revisions**

*In the previous `kubectl rollout history` step, you got a list of DaemonSet revisions. Each revision is stored in a resource named `ControllerRevision`.*

*To see what is stored in each revision, find the DaemonSet revision raw resources:*

```
kubectl get controllerrevision -l <daemonset-selector-key>=<daemonset-selector-value>
```

*This returns a list of ControllerRevisions:*

NAME	CONTROLLER	REVISION	AGE
<daemonset-name>-<revision-hash>	DaemonSet/<daemonset-name>	1	1h
<daemonset-name>-<revision-hash>	DaemonSet/<daemonset-name>	2	1h

*Each ControllerRevision stores the annotations and template of a DaemonSet revision.*

*`kubectl rollout undo` takes a specific ControllerRevision and replaces DaemonSet template with the template stored in the ControllerRevision. `kubectl rollout undo` is equivalent to updating DaemonSet template to a previous revision through other commands, such as `kubectl edit` or `kubectl apply`.*

**Note:** DaemonSet revisions only roll forward. That is to say, after a rollback completes, the revision number (`.revision` field) of the ControllerRevision being rolled back to will advance. For example, if you have revision 1 and 2 in the system, and roll back from revision 2 to revision 1, the ControllerRevision with `.revision: 1` will become `.revision: 3`.

## Troubleshooting

- See [troubleshooting DaemonSet rolling update](#).

## 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](#).

Last modified May 20, 2020 at 8:53 AM PST: [Revise DaemonSet rollback task \(82eb6672e\)](#)

[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Performing a rollback on a DaemonSet](#)
  - [Step 1: Find the DaemonSet revision you want to roll back to](#)
  - [Step 2: Roll back to a specific revision](#)
  - [Step 3: Watch the progress of the DaemonSet rollback](#)
- [Understanding DaemonSet revisions](#)
- [Troubleshooting](#)

# Service Catalog

Install the Service Catalog extension API.

---

[Install Service Catalog using Helm](#)

[Install Service Catalog using SC](#)

## Install Service Catalog using Helm

Service Catalog is an extension API that enables applications running in Kubernetes clusters to easily use external managed software offerings, such as a datastore service offered by a cloud provider.

It provides a way to list, provision, and bind with external [Managed Services](#) from [Service Brokers](#) without needing detailed knowledge about how those services are created or managed.

Use [Helm](#) to install Service Catalog on your Kubernetes cluster. Up to date information on this process can be found at the [kubernetes-sigs/service-catalog](#) repo.

## Before you begin

- Understand the key concepts of [Service Catalog](#).
- Service Catalog requires a Kubernetes cluster running version 1.7 or higher.
- You must have a Kubernetes cluster with cluster DNS enabled.
  - If you are using a cloud-based Kubernetes cluster or [Minikube](#), you may already have cluster DNS enabled.

- If you are using `hack/local-up-cluster.sh`, ensure that the `KUBE_ENABLE_CLUSTER_DNS` environment variable is set, then run the `install` script.
- [Install and setup kubectl](#) v1.7 or higher. Make sure it is configured to connect to the Kubernetes cluster.
- Install [Helm](#) v2.7.0 or newer.
  - Follow the [Helm install instructions](#).
  - If you already have an appropriate version of Helm installed, execute `helm init` to install Tiller, the server-side component of Helm.

## Add the service-catalog Helm repository

Once Helm is installed, add the service-catalog Helm repository to your local machine by executing the following command:

```
helm repo add svc-cat https://svc-catalog-charts.storage.googleapis.com
```

Check to make sure that it installed successfully by executing the following command:

```
helm search repo service-catalog
```

If the installation was successful, the command should output the following:

NAME	CHART VERSION	APP VERSION
DESCRIPTION		
svc-cat/catalog	0.2.1	
service-catalog API server and controller-manager helm chart		
svc-cat/catalog-v0.2	0.2.2	
service-catalog API server and controller-manager helm chart		

## Enable RBAC

Your Kubernetes cluster must have RBAC enabled, which requires your Tiller Pod(s) to have `cluster-admin` access.

When using Minikube v0.25 or older, you must run Minikube with RBAC explicitly enabled:

```
minikube start --extra-config=apiserver.Authorization.Mode=RBAC
```

When using Minikube v0.26+, run:

```
minikube start
```

With Minikube v0.26+, do not specify `--extra-config`. The flag has since been changed to `--extra-config=apiserver.authorization-mode` and

*Minikube now uses RBAC by default. Specifying the older flag may cause the start command to hang.*

*If you are using `hack/local-up-cluster.sh`, set the `AUTHORIZATION_MODE` environment variable with the following values:*

```
AUTHORIZATION_MODE=Node,RBAC hack/local-up-cluster.sh -0
```

*By default, `helm init` installs the Tiller Pod into the `kube-system` namespace, with Tiller configured to use the `default` service account.*

**Note:** If you used the `--tiller-namespace` or `--service-account` flags when running `helm init`, the `--serviceaccount` flag in the following command needs to be adjusted to reference the appropriate namespace and ServiceAccount name.

*Configure Tiller to have `cluster-admin` access:*

```
kubectl create clusterrolebinding tiller-cluster-admin \
  --clusterrole=cluster-admin \
  --serviceaccount=kube-system:default
```

## **Install Service Catalog in your Kubernetes cluster**

*Install Service Catalog from the root of the Helm repository using the following command:*

- [Helm version 3](#)
- [Helm version 2](#)

```
helm install catalog svc-cat/catalog --namespace catalog
```

```
helm install svc-cat/catalog --name catalog --namespace catalog
```

## **What's next**

- View [sample service brokers](#).
- Explore the [kubernetes-sigs/service-catalog](#) project.

## **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](#).

Last modified September 30, 2020 at 10:44 PM PST: [Fix missing command in helm search statement \(6fe4034d8\)](#)

[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Add the service-catalog Helm repository](#)
- [Enable RBAC](#)
- [Install Service Catalog in your Kubernetes cluster](#)
- [What's next](#)

# Install Service Catalog using SC

*Service Catalog is an extension API that enables applications running in Kubernetes clusters to easily use external managed software offerings, such as a datastore service offered by a cloud provider.*

*It provides a way to list, provision, and bind with external [Managed Services](#) from [Service Brokers](#) without needing detailed knowledge about how those services are created or managed.*

*You can use the GCP [Service Catalog Installer](#) tool to easily install or uninstall Service Catalog on your Kubernetes cluster, linking it to Google Cloud projects.*

*Service Catalog itself can work with any kind of managed service, not just Google Cloud.*

## Before you begin

- Understand the key concepts of [Service Catalog](#).
- Install [Go 1.6+](#) and set the GOPATH.
- Install the [cfssl](#) tool needed for generating SSL artifacts.
- Service Catalog requires Kubernetes version 1.7+.
- [Install and setup kubectl](#) so that it is configured to connect to a Kubernetes v1.7+ cluster.
- The kubectl user must be bound to the cluster-admin role for it to install Service Catalog. To ensure that this is true, run the following command:

```
kubectl create clusterrolebinding cluster-admin-binding --clusterrole=cluster-admin --user=<user-name>
```

# **Install sc in your local environment**

The installer runs on your local computer as a CLI tool named *sc*.

Install using *go get*:

```
go get github.com/GoogleCloudPlatform/k8s-service-catalog/
installer/cmd/sc
```

*sc* should now be installed in your *GOPATH/bin* directory.

# **Install Service Catalog in your Kubernetes cluster**

First, verify that all dependencies have been installed. Run:

```
sc check
```

If the check is successful, it should return:

```
Dependency check passed. You are good to go.
```

Next, run the *install* command and specify the *storageclass* that you want to use for the backup:

```
sc install --etcd-backup-storageclass "standard"
```

# **Uninstall Service Catalog**

If you would like to uninstall Service Catalog from your Kubernetes cluster using the *sc* tool, run:

```
sc uninstall
```

## **What's next**

- View [sample service brokers](#).
- Explore the [kubernetes-sigs/service-catalog](#) project.

## **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](#).

Last modified October 17, 2020 at 3:21 PM PST: [update kubernetes-incubator references \(a8b6551c2\)](#)

[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Install sc in your local environment](#)
- [Install Service Catalog in your Kubernetes cluster](#)
- [Uninstall Service Catalog](#)
- [What's next](#)

# Networking

Learn how to configure networking for your cluster.

---

[Validate IPv4/IPv6 dual-stack](#)

## Validate IPv4/IPv6 dual-stack

This document shares how to validate IPv4/IPv6 dual-stack enabled Kubernetes clusters.

### Before you begin

- Provider support for dual-stack networking (Cloud provider or otherwise must be able to provide Kubernetes nodes with routable IPv4/IPv6 network interfaces)
- A [network plugin](#) that supports dual-stack (such as Kubelet or Calico)
- [Dual-stack enabled](#) cluster

Your Kubernetes server must be version v1.20. To check the version, enter `kubectl version`.

### Validate addressing

#### Validate node addressing

Each dual-stack Node should have a single IPv4 block and a single IPv6 block allocated. Validate that IPv4/IPv6 Pod address ranges are configured by running the following command. Replace the sample node name with a valid dual-stack Node from your cluster. In this example, the Node's name is `k8s-linuxpath1-34450317-0`:

```
kubectl get nodes k8s-linuxpath1-34450317-0 -o go-template --template='{{range .spec.podCIDRs}}{{printf "%s\n" .}}{{end}}'
```

```
10.244.1.0/24  
a00:100::/24
```

There should be one IPv4 block and one IPv6 block allocated.

Validate that the node has an IPv4 and IPv6 interface detected (replace node name with a valid node from the cluster. In this example the node name is k8s-linuxpool1-34450317-0):

```
kubectl get nodes k8s-linuxpool1-34450317-0 -o go-template --template='{{range .status.addresses}}{{printf "%s: %s\n" .type .address}}{{end}}'
```

```
Hostname: k8s-linuxpool1-34450317-0  
InternalIP: 10.240.0.5  
InternalIP: 2001:1234:5678:9abc::5
```

## Validate Pod addressing

Validate that a Pod has an IPv4 and IPv6 address assigned. (replace the Pod name with a valid Pod in your cluster. In this example the Pod name is pod01)

```
kubectl get pods pod01 -o go-template --template='{{range .status.podIPs}}{{printf "%s\n" .ip}}{{end}}'
```

```
10.244.1.4  
a00:100::4
```

You can also validate Pod IPs using the Downward API via the `status.podIPs` fieldPath. The following snippet demonstrates how you can expose the Pod IPs via an environment variable called `MY_POD_IPS` within a container.

```
env:  
- name: MY_POD_IPS  
  valueFrom:  
    fieldRef:  
      fieldPath: status.podIPs
```

The following command prints the value of the `MY_POD_IPS` environment variable from within a container. The value is a comma separated list that corresponds to the Pod's IPv4 and IPv6 addresses.

```
kubectl exec -it pod01 -- set | grep MY_POD_IPS
```

```
MY_POD_IPS=10.244.1.4,a00:100::4
```

The Pod's IP addresses will also be written to `/etc/hosts` within a container. The following command executes a `cat` on `/etc/hosts` on a dual stack Pod. From the output you can verify both the IPv4 and IPv6 IP address for the Pod.

```
kubectl exec -it pod01 -- cat /etc/hosts
```

```
# Kubernetes-managed hosts file.  
127.0.0.1      localhost  
::1            localhost ip6-localhost ip6-loopback  
fe00::0        ip6-localnet  
fe00::0        ip6-mcastprefix  
fe00::1        ip6-allnodes  
fe00::2        ip6-allrouters  
10.244.1.4    pod01  
a00:100::4    pod01
```

## Validate Services

Create the following Service that does not explicitly define `.spec.ipFamilyPolicy`. Kubernetes will assign a cluster IP for the Service from the first configured `service-cluster-ip-range` and set the `.spec.ipFamilyPolicy` to `SingleStack`.

[service/networking/dual-stack-default-svc.yaml](#)



```
apiVersion: v1  
kind: Service  
metadata:  
  name: my-service  
  labels:  
    app: MyApp  
spec:  
  selector:  
    app: MyApp  
  ports:  
  - protocol: TCP  
    port: 80
```

Use `kubectl` to view the YAML for the Service.

```
kubectl get svc my-service -o yaml
```

The Service has `.spec.ipFamilyPolicy` set to `SingleStack` and `.spec.clusterIP` set to an IPv4 address from the first configured range set via `--service-cluster-ip-range` flag on kube-controller-manager.

```
apiVersion: v1
kind: Service
metadata:
  name: my-service
  namespace: default
spec:
  clusterIP: 10.0.217.164
  clusterIPs:
  - 10.0.217.164
  ipFamilies:
  - IPv4
  ipFamilyPolicy: SingleStack
  ports:
  - port: 80
    protocol: TCP
    targetPort: 9376
  selector:
    app: MyApp
  sessionAffinity: None
  type: ClusterIP
status:
  loadBalancer: {}
```

Create the following Service that explicitly defines IPv6 as the first array element in `.spec.ipFamilies`. Kubernetes will assign a cluster IP for the Service from the IPv6 range configured `service-cluster-ip-range` and set the `.spec.ipFamilyPolicy` to `SingleStack`.

[service/networking/dual-stack-ipfamilies-ipv6.yaml](#)  


```
apiVersion: v1
kind: Service
metadata:
  name: my-service
  labels:
    app: MyApp
spec:
  ipFamilies:
  - IPv6
  selector:
    app: MyApp
```

```
ports:
  - protocol: TCP
    port: 80
```

Use `kubectl` to view the YAML for the Service.

```
kubectl get svc my-service -o yaml
```

The Service has `.spec.ipFamilyPolicy` set to `SingleStack` and `.spec.clusterIP` set to an IPv6 address from the IPv6 range set via `--service-cluster-ip-range` flag on `kube-controller-manager`.

```
apiVersion: v1
kind: Service
metadata:
  labels:
    app: MyApp
    name: my-service
spec:
  clusterIP: fd00::5118
  clusterIPs:
  - fd00::5118
  ipFamilies:
  - IPv6
  ipFamilyPolicy: SingleStack
  ports:
  - port: 80
    protocol: TCP
    targetPort: 80
  selector:
    app: MyApp
  sessionAffinity: None
  type: ClusterIP
status:
  loadBalancer: {}
```

Create the following Service that explicitly defines `PreferDualStack` in `.spec.ipFamilyPolicy`. Kubernetes will assign both IPv4 and IPv6 addresses (as this cluster has dual-stack enabled) and select the `.spec.ClusterIP` from the list of `.spec.ClusterIPs` based on the address family of the first element in the `.spec.ipFamilies` array.

[service/networking/dual-stack-preferred-svc.yaml](#)



```
apiVersion: v1
kind: Service
metadata:
```

```

name: my-service
labels:
  app: MyApp
spec:
  ipFamilyPolicy: PreferDualStack
  selector:
    app: MyApp
  ports:
    - protocol: TCP
      port: 80

```

**Note:**

The `kubectl get svc` command will only show the primary IP in the `CLUSTER-IP` field.

```
kubectl get svc -l app=MyApp
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP
IP	PORT(S)	AGE	
my-service	ClusterIP	10.0.216.242	
<none>	80/TCP	5s	

Validate that the Service gets cluster IPs from the IPv4 and IPv6 address blocks using `kubectl describe`. You may then validate access to the service via the IPs and ports.

```
kubectl describe svc -l app=MyApp
```

Name:	my-service
Namespace:	default
Labels:	app=MyApp
Annotations:	<none>
Selector:	app=MyApp
Type:	ClusterIP
IP Family Policy:	PreferDualStack
IP Families:	IPv4,IPv6
IP:	10.0.216.242
IPs:	10.0.216.242, fd00::af55
Port:	<unset> 80/TCP
TargetPort:	9376/TCP
Endpoints:	<none>
Session Affinity:	None
Events:	<none>

## Create a dual-stack load balanced Service

If the cloud provider supports the provisioning of IPv6 enabled external load balancers, create the following Service with `PreferDualStack` in `.spec.ipFamilyPolicy`, `IPv6` as the first element of the `.spec.ipFamilies` array and the `type` field set to `LoadBalancer`.

[service/networking/dual-stack-prefer-ipv6-lb-svc.yaml](#)

```
apiVersion: v1
kind: Service
metadata:
  name: my-service
  labels:
    app: MyApp
spec:
  ipFamilyPolicy: PreferDualStack
  ipFamilies:
    - IPv6
  type: LoadBalancer
  selector:
    app: MyApp
  ports:
    - protocol: TCP
      port: 80
```

Check the Service:

```
kubectl get svc -l app=MyApp
```

Validate that the Service receives a `CLUSTER-IP` address from the `IPv6` address block along with an `EXTERNAL-IP`. You may then validate access to the service via the IP and port.

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP
PORT(S)	AGE		
my-service	LoadBalancer	fd00::7ebc	2603:1030:805::5
80:30790/TCP	35s		

## 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](#).

Last modified October 26, 2020 at 1:06 PM PST: [Dual-stack docs for Kubernetes 1.20 \(8a3244fdd\)](#)

[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Validate addressing](#)
  - [Validate node addressing](#)
  - [Validate Pod addressing](#)
- [Validate Services](#)
  - [Create a dual-stack load balanced Service](#)

# Configure a **kubelet** image credential provider

Configure the kubelet's image credential provider plugin

**FEATURE STATE:** Kubernetes v1.20 [alpha]

Starting from Kubernetes v1.20, the kubelet can dynamically retrieve credentials for a container image registry using exec plugins. The kubelet and the exec plugin communicate through stdio (stdin, stdout, and stderr) using Kubernetes versioned APIs. These plugins allow the kubelet to request credentials for a container registry dynamically as opposed to storing static credentials on disk. For example, the plugin may talk to a local metadata server to retrieve short-lived credentials for an image that is being pulled by the kubelet.

You may be interested in using this capability if any of the below are true:

- API calls to a cloud provider service are required to retrieve authentication information for a registry.
- Credentials have short expiration times and requesting new credentials frequently is required.
- Storing registry credentials on disk or in imagePullSecrets is not acceptable.

This guide demonstrates how to configure the kubelet's image credential provider plugin mechanism.

## Before you begin

- The kubelet image credential provider is introduced in v1.20 as an alpha feature. As with other alpha features, a feature gate `Kubelet CredentialProviders` must be enabled on only the kubelet for the feature to work.

- A working implementation of a credential provider exec plugin. You can build your own plugin or use one provided by cloud providers.

## **Installing Plugins on Nodes**

A credential provider plugin is an executable binary that will be run by the kubelet. Ensure that the plugin binary exists on every node in your cluster and stored in a known directory. The directory will be required later when configuring kubelet flags.

## **Configuring the Kubelet**

In order to use this feature, the kubelet expects two flags to be set:

- `--image-credential-provider-config` - the path to the credential provider plugin config file.
- `--image-credential-provider-bin-dir` - the path to the directory where credential provider binaries are located.

### **Configure a kubelet credential provider**

The configuration file passed into `--image-credential-provider-config` is read by the kubelet to determine which exec plugins should be invoked for which container images. Here's an example configuration file you may end up using if you are using the [ECR](#)-based plugin:

```

kind: CredentialProviderConfig
apiVersion: kubelet.config.k8s.io/v1alpha1
# providers is a list of credential provider plugins that
will be enabled by the kubelet.
# Multiple providers may match against a single image, in
which case credentials
# from all providers will be returned to the kubelet. If
multiple providers are called
# for a single image, the results are combined. If providers
return overlapping
# auth keys, the value from the provider earlier in this
list is used.
providers:
  # name is the required name of the credential provider. It
  must match the name of the
    # provider executable as seen by the kubelet. The
    executable must be in the kubelet's
      # bin directory (set by the --image-credential-provider-
      bin-dir flag).
    - name: ecr
      # matchImages is a required list of strings used to
      match against images in order to
        # determine if this provider should be invoked. If one

```

```

of the strings matches the
    # requested image from the kubelet, the plugin will be
invoked and given a chance
        # to provide credentials. Images are expected to contain
the registry domain
        # and URL path.
        #
        # Each entry in matchImages is a pattern which can
optionally contain a port and a path.
        # Globs can be used in the domain, but not in the port
or the path. Globs are supported
        # as subdomains like '*.k8s.io' or 'k8s.*.io', and top-
level-domains such as 'k8s.*'.
        # Matching partial subdomains like 'app*.k8s.io' is also
supported. Each glob can only match
            # a single subdomain segment, so *.io does not match
*.k8s.io.
        #
        # A match exists between an image and a matchImage when
all of the below are true:
            # - Both contain the same number of domain parts and
each part matches.
            # - The URL path of an imageMatch must be a prefix of
the target image URL path.
            # - If the imageMatch contains a port, then the port
must match in the image as well.
        #
        # Example values of matchImages:
        # - 123456789.dkr.ecr.us-east-1.amazonaws.com
        # - *.azurecr.io
        # - gcr.io
        # - *.*.registry.io
        # - registry.io:8080/path
matchImages:
- "*.dkr.ecr.*.amazonaws.com"
- "*.dkr.ecr.*.amazonaws.cn"
- "*.dkr.ecr-fips.*.amazonaws.com"
- "*.dkr.ecr.us-iso-east-1.c2s.ic.gov"
- "*.dkr.ecr.us-isob-east-1.sc2s.sgov.gov"
    # defaultCacheDuration is the default duration the
plugin will cache credentials in-memory
    # if a cache duration is not provided in the plugin
response. This field is required.
defaultCacheDuration: "12h"
    # Required input version of the exec
CredentialProviderRequest. The returned
CredentialProviderResponse
    # MUST use the same encoding version as the input.
Current supported values are:
    # - credentialprovider.kubelet.k8s.io/v1alpha1
apiVersion: credentialprovider.kubelet.k8s.io/v1alpha1
    # Arguments to pass to the command when executing it.

```

```

# +optional
args:
- get-credentials
# Env defines additional environment variables to expose
to the process. These
# are unioned with the host's environment, as well as
variables client-go uses
# to pass argument to the plugin.
# +optional
env:
- name: AWS_PROFILE
  value: example_profile

```

*The providers field is a list of enabled plugins used by the kubelet. Each entry has a few required fields:*

- *name*: the name of the plugin which MUST match the name of the executable binary that exists in the directory passed into `--image-credential-provider-bin-dir`.
- *matchImages*: a list of strings used to match against images in order to determine if this provider should be invoked. More on this below.
- *defaultCacheDuration*: the default duration the kubelet will cache credentials in-memory if a cache duration was not specified by the plugin.
- *apiVersion*: the api version that the kubelet and the exec plugin will use when communicating.

*Each credential provider can also be given optional args and environment variables as well. Consult the plugin implementors to determine what set of arguments and environment variables are required for a given plugin.*

## **Configure image matching**

*The matchImages field for each credential provider is used by the kubelet to determine whether a plugin should be invoked for a given image that a Pod is using. Each entry in matchImages is an image pattern which can optionally contain a port and a path. Globs can be used in the domain, but not in the port or the path. Globs are supported as subdomains like `*.k8s.io` or `k8s.*.io`, and top-level domains such as `k8s.*`. Matching partial subdomains like `app*.k8s.io` is also supported. Each glob can only match a single subdomain segment, so `*.io` does NOT match `*.k8s.io`.*

*A match exists between an image name and a matchImage entry when all of the below are true:*

- Both contain the same number of domain parts and each part matches.
- The URL path of match image must be a prefix of the target image URL path.

- If the `imageMatch` contains a port, then the port must match in the image as well.

Some example values of `matchImages` patterns are:

- `123456789.dkr.ecr.us-east-1.amazonaws.com`
- `*.azurecr.io`
- `gcr.io`
- `*.*.registry.io`
- `foo.registry.io:8080/path`

## 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](#).

Last modified November 06, 2020 at 10:17 AM PST: [Add docs for configuring kubelet credential provider plugins \(dbdde629c\)](#)

[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Installing Plugins on Nodes](#)
- [Configuring the Kubelet](#)
  - [Configure a kubelet credential provider](#)

# Extend kubectl with plugins

Extend kubectl by creating and installing kubectl plugins.

This guide demonstrates how to install and write extensions for [kubectl](#). By thinking of core kubectl commands as essential building blocks for interacting with a Kubernetes cluster, a cluster administrator can think of plugins as a means of utilizing these building blocks to create more complex behavior. Plugins extend kubectl with new sub-commands, allowing for new and custom features not included in the main distribution of kubectl.

## Before you begin

You need to have a working kubectl binary installed.

# **Installing kubectl plugins**

*A plugin is nothing more than a standalone executable file, whose name begins with `kubectl-`. To install a plugin, simply move its executable file to anywhere on your PATH.*

*You can also discover and install kubectl plugins available in the open source using [Krew](#). Krew is a plugin manager maintained by the Kubernetes SIG CLI community.*

**Caution:** Kubectl plugins available via the [Krew plugin index](#) are not audited for security. You should install and run third-party plugins at your own risk, since they are arbitrary programs running on your machine.

## **Discovering plugins**

*kubectl provides a command `kubectl plugin list` that searches your PATH for valid plugin executables. Executing this command causes a traversal of all files in your PATH. Any files that are executable, and begin with `kubectl-` will show up in the order in which they are present in your PATH in this command's output. A warning will be included for any files beginning with `kubectl-` that are not executable. A warning will also be included for any valid plugin files that overlap each other's name.*

*You can use [Krew](#) to discover and install kubectl plugins from a community-curated [plugin index](#).*

## **Limitations**

*It is currently not possible to create plugins that overwrite existing kubectl commands. For example, creating a plugin `kubectl-version` will cause that plugin to never be executed, as the existing `kubectl version` command will always take precedence over it. Due to this limitation, it is also not possible to use plugins to add new subcommands to existing kubectl commands. For example, adding a subcommand `kubectl create foo` by naming your plugin `kubectl-create-foo` will cause that plugin to be ignored.*

*`kubectl plugin list` shows warnings for any valid plugins that attempt to do this.*

## **Writing kubectl plugins**

*You can write a plugin in any programming language or script that allows you to write command-line commands.*

*There is no plugin installation or pre-loading required. Plugin executables receive the inherited environment from the `kubectl` binary. A plugin determines which command path it wishes to implement based*

on its name. For example, a plugin wanting to provide a new command `kubectl foo`, would simply be named `kubectl-foo`, and live somewhere in your PATH.

## Example plugin

```
#!/bin/bash

# optional argument handling
if [[ "$1" == "version" ]]
then
    echo "1.0.0"
    exit 0
fi

# optional argument handling
if [[ "$1" == "config" ]]
then
    echo "$KUBECONFIG"
    exit 0
fi

echo "I am a plugin named kubectl-foo"
```

## Using a plugin

To use the above plugin, simply make it executable:

```
sudo chmod +x ./kubectl-foo
```

and place it anywhere in your PATH:

```
sudo mv ./kubectl-foo /usr/local/bin
```

You may now invoke your plugin as a `kubectl` command:

```
kubectl foo
```

```
I am a plugin named kubectl-foo
```

All args and flags are passed as-is to the executable:

```
kubectl foo version
```

```
1.0.0
```

All environment variables are also passed as-is to the executable:

```
export KUBECONFIG=~/.kube/config
kubectl foo config
```

```
/home/<user>/~/.kube/config
```

```
KUBECONFIG=/etc/kube/config kubectl foo config  
/etc/kube/config
```

*Additionally, the first argument that is passed to a plugin will always be the full path to the location where it was invoked (\$0 would equal /usr/local/bin/kubectl-foo in the example above).*

## Naming a plugin

*As seen in the example above, a plugin determines the command path that it will implement based on its filename. Every sub-command in the command path that a plugin targets, is separated by a dash (-). For example, a plugin that wishes to be invoked whenever the command kubectl foo bar baz is invoked by the user, would have the filename of kubectl-foo-bar-baz.*

## Flags and argument handling

### Note:

*The plugin mechanism does not create any custom, plugin-specific values or environment variables for a plugin process.*

*An older kubectl plugin mechanism provided environment variables such as KUBECTL\_PLUGINS\_CURRENT\_NAMESPACE; that no longer happens.*

*kubectl plugins must parse and validate all of the arguments passed to them. See [using the command line runtime package](#) for details of a Go library aimed at plugin authors.*

*Here are some additional cases where users invoke your plugin while providing additional flags and arguments. This builds upon the kubectl-foo-bar-baz plugin from the scenario above.*

*If you run kubectl foo bar baz arg1 --flag=value arg2, kubectl's plugin mechanism will first try to find the plugin with the longest possible name, which in this case would be kubectl-foo-bar-baz-arg1. Upon not finding that plugin, kubectl then treats the last dash-separated value as an argument (arg1 in this case), and attempts to find the next longest possible name, kubectl-foo-bar-baz. Upon having found a plugin with this name, kubectl then invokes that plugin, passing all args and flags after the plugin's name as arguments to the plugin process.*

*Example:*

```
# create a plugin  
echo -e '#!/bin/bash\n\n#echo "My first command-line argument  
was $1"' > kubectl-foo-bar-baz
```

```
sudo chmod +x ./kubectl-foo-bar-baz  
# "install" your plugin by moving it to a directory in your  
$PATH  
sudo mv ./kubectl-foo-bar-baz /usr/local/bin
```

```
# check that kubectl recognizes your plugin  
kubectl plugin list
```

The following kubectl-compatible plugins are available:

```
/usr/local/bin/kubectl-foo-bar-baz
```

```
# test that calling your plugin via a "kubectl" command works  
# even when additional arguments and flags are passed to your  
# plugin executable by the user.  
kubectl foo bar baz arg1 --meaningless-flag=true
```

My first command-line argument was arg1

As you can see, your plugin was found based on the kubectl command specified by a user, and all extra arguments and flags were passed as-is to the plugin executable once it was found.

### **Names with dashes and underscores**

Although the kubectl plugin mechanism uses the dash (-) in plugin filenames to separate the sequence of sub-commands processed by the plugin, it is still possible to create a plugin command containing dashes in its commandline invocation by using underscores (\_) in its filename.

Example:

```
# create a plugin containing an underscore in its filename  
echo -e '#!/bin/bash\n\nnecho "I am a plugin with a dash in  
my name"' > ./kubectl-foo_bar  
sudo chmod +x ./kubectl-foo_bar  
  
# move the plugin into your $PATH  
sudo mv ./kubectl-foo_bar /usr/local/bin  
  
# You can now invoke your plugin via kubectl:  
kubectl foo-bar
```

I am a plugin with a dash in my name

Note that the introduction of underscores to a plugin filename does not prevent you from having commands such as kubectl foo\_bar. The command from the above example, can be invoked using either a dash (-) or an underscore (\_):

```
# You can invoke your custom command with a dash  
kubectl foo-bar
```

*I am a plugin with a dash in my name*

```
# You can also invoke your custom command with an underscore  
kubectl foo_bar
```

*I am a plugin with a dash in my name*

### **Name conflicts and overshadowing**

*It is possible to have multiple plugins with the same filename in different locations throughout your PATH. For example, given a PATH with the following value: PATH=/usr/local/bin/plugins:/usr/local/bin/moreplugins, a copy of plugin kubectl-foo could exist in /usr/local/bin/plugins and /usr/local/bin/moreplugins, such that the output of the kubectl plugin list command is:*

```
PATH=/usr/local/bin/plugins:/usr/local/bin/moreplugins  
kubectl plugin list
```

*The following kubectl-compatible plugins are available:*

```
/usr/local/bin/plugins/kubectl-foo  
/usr/local/bin/moreplugins/kubectl-foo  
    - warning: /usr/local/bin/moreplugins/kubectl-foo is  
overshadowed by a similarly named plugin: /usr/local/bin/  
plugins/kubectl-foo  
  
error: one plugin warning was found
```

*In the above scenario, the warning under /usr/local/bin/moreplugins/kubectl-foo tells you that this plugin will never be executed. Instead, the executable that appears first in your PATH, /usr/local/bin/plugins/kubectl-foo, will always be found and executed first by the kubectl plugin mechanism.*

*A way to resolve this issue is to ensure that the location of the plugin that you wish to use with kubectl always comes first in your PATH. For example, if you want to always use /usr/local/bin/moreplugins/kubectl-foo anytime that the kubectl command kubectl foo was invoked, change the value of your PATH to be /usr/local/bin/moreplugins:/usr/local/bin/plugins.*

### **Invocation of the longest executable filename**

*There is another kind of overshadowing that can occur with plugin filenames. Given two plugins present in a user's PATH: kubectl-foo-bar and kubectl-foo-bar-baz, the kubectl plugin mechanism will always choose the longest possible plugin name for a given user command. Some examples below, clarify this further:*

```
# for a given kubectl command, the plugin with the longest
possible filename will always be preferred
kubectl foo bar baz
```

*Plugin kubectl-foo-bar-baz is executed*

```
kubectl foo bar
```

*Plugin kubectl-foo-bar is executed*

```
kubectl foo bar baz buz
```

*Plugin kubectl-foo-bar-baz is executed, with "buz" as its
first argument*

```
kubectl foo bar buz
```

*Plugin kubectl-foo-bar is executed, with "buz" as its first
argument*

*This design choice ensures that plugin sub-commands can be
implemented across multiple files, if needed, and that these sub-
commands can be nested under a "parent" plugin command:*

```
ls ./plugin_command_tree
```

```
kubectl-parent
kubectl-parent-subcommand
kubectl-parent-subcommand-subsubcommand
```

## **Checking for plugin warnings**

*You can use the aforementioned kubectl plugin list command to
ensure that your plugin is visible by kubectl, and verify that there are
no warnings preventing it from being called as a kubectl command.*

```
kubectl plugin list
```

*The following kubectl-compatible plugins are available:*

```
test/fixtures/pkg/kubectl/plugins/kubectl-foo
/usr/local/bin/kubectl-foo
    - warning: /usr/local/bin/kubectl-foo is overshadowed by a
similarly named plugin: test/fixtures/pkg/kubectl/plugins/
kubectl-foo
plugins/kubectl-invalid
    - warning: plugins/kubectl-invalid identified as a kubectl
plugin, but it is not executable
```

*error: 2 plugin warnings were found*

## **Using the command line runtime package**

If you're writing a plugin for `kubectl` and you're using Go, you can make use of the [cli-runtime](#) utility libraries.

These libraries provide helpers for parsing or updating a user's [kubeconfig](#) file, for making REST-style requests to the API server, or to bind flags associated with configuration and printing.

See the [Sample CLI Plugin](#) for an example usage of the tools provided in the CLI Runtime repo.

## **Distributing kubectl plugins**

If you have developed a plugin for others to use, you should consider how you package it, distribute it and deliver updates to your users.

### **Krew**

[Krew](#) offers a cross-platform way to package and distribute your plugins. This way, you use a single packaging format for all target platforms (Linux, Windows, macOS etc) and deliver updates to your users. Krew also maintains a [plugin index](#) so that other people can discover your plugin and install it.

### **Native / platform specific package management**

Alternatively, you can use traditional package managers such as, `apt` or `yum` on Linux, Chocolatey on Windows, and Homebrew on macOS. Any package manager will be suitable if it can place new executables placed somewhere in the user's `PATH`. As a plugin author, if you pick this option then you also have the burden of updating your `kubectl` plugin's distribution package across multiple platforms for each release.

### **Source code**

You can publish the source code; for example, as a Git repository. If you choose this option, someone who wants to use that plugin must fetch the code, set up a build environment (if it needs compiling), and deploy the plugin. If you also make compiled packages available, or use Krew, that will make installs easier.

## **What's next**

- Check the Sample CLI Plugin repository for a [detailed example](#) of a plugin written in Go. In case of any questions, feel free to reach out to the [SIG CLI team](#).
- Read about [Krew](#), a package manager for `kubectl` plugins.

# 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](#).

Last modified November 18, 2020 at 10:59 PM PST: [Update the URL of krew plugin list \(52a54dbf0\)](#)

[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [Installing kubectl plugins](#)
  - [Discovering plugins](#)
- [Writing kubectl plugins](#)
  - [Example plugin](#)
  - [Using a plugin](#)
  - [Naming a plugin](#)
  - [Checking for plugin warnings](#)
  - [Using the command line runtime package](#)
- [Distributing kubectl plugins](#)
  - [Krew](#)
  - [Native / platform specific package management](#)
  - [Source code](#)
- [What's next](#)

# Manage HugePages

Configure and manage huge pages as a schedulable resource in a cluster.

**FEATURE STATE:** Kubernetes v1.20 [stable]

Kubernetes supports the allocation and consumption of pre-allocated huge pages by applications in a Pod. This page describes how users can consume huge pages.

## Before you begin

1. Kubernetes nodes must pre-allocate huge pages in order for the node to report its huge page capacity. A node can pre-allocate huge pages for multiple sizes.

The nodes will automatically discover and report all huge page resources as schedulable resources.

# API

Huge pages can be consumed via container level resource requirements using the resource name `hugepages-<size>`, where `<size>` is the most compact binary notation using integer values supported on a particular node. For example, if a node supports 2048KiB and 1048576KiB page sizes, it will expose a schedulable resources `hugepages-2Mi` and `hugepages-1Gi`. Unlike CPU or memory, huge pages do not support overcommit. Note that when requesting hugepage resources, either memory or CPU resources must be requested as well.

A pod may consume multiple huge page sizes in a single pod spec. In this case it must use `medium: HugePages-<hugepagesize>` notation for all volume mounts.

```
apiVersion: v1
kind: Pod
metadata:
  name: huge-pages-example
spec:
  containers:
    - name: example
      image: fedora:latest
      command:
        - sleep
        - inf
      volumeMounts:
        - mountPath: /hugepages-2Mi
          name: hugepage-2mi
        - mountPath: /hugepages-1Gi
          name: hugepage-1gi
      resources:
        limits:
          hugepages-2Mi: 100Mi
          hugepages-1Gi: 2Gi
          memory: 100Mi
        requests:
          memory: 100Mi
  volumes:
    - name: hugepage-2mi
      emptyDir:
        medium: HugePages-2Mi
    - name: hugepage-1gi
      emptyDir:
        medium: HugePages-1Gi
```

A pod may use `medium: HugePages` only if it requests huge pages of one size.

```
apiVersion: v1
kind: Pod
metadata:
```

```

  name: huge-pages-example
spec:
  containers:
    - name: example
      image: fedora:latest
      command:
        - sleep
        - inf
      volumeMounts:
        - mountPath: /hugepages
          name: hugepage
      resources:
        limits:
          hugepages-2Mi: 100Mi
          memory: 100Mi
        requests:
          memory: 100Mi
  volumes:
    - name: hugepage
      emptyDir:
        medium: HugePages

```

- Huge page requests must equal the limits. This is the default if limits are specified, but requests are not.
- Huge pages are isolated at a container scope, so each container has own limit on their cgroup sandbox as requested in a container spec.
- EmptyDir volumes backed by huge pages may not consume more huge page memory than the pod request.
- Applications that consume huge pages via `shmget()` with `SHM_HUGE_TLB` must run with a supplemental group that matches `proc/sys/vm/hugetlb_shm_group`.
- Huge page usage in a namespace is controllable via ResourceQuota similar to other compute resources like `cpu` or `memory` using the `hugepages-<size>` token.
- Support of multiple sizes huge pages is feature gated. It can be disabled with the `HugePageStorageMediumSize` [feature gate](#) on the [kubelet](#) and [kube-apiserver](#) (`--feature-gates=HugePageStorageMediumSize=true`).

## 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](#).

Last modified June 16, 2020 at 3:26 AM PST: [Remove statements about the future \(6df756c35\)](#)

[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Before you begin](#)
- [API](#)

# Schedule GPUs

Configure and schedule GPUs for use as a resource by nodes in a cluster.

**FEATURE STATE:** Kubernetes v1.10 [beta]

Kubernetes includes **experimental** support for managing AMD and NVIDIA GPUs (graphical processing units) across several nodes.

This page describes how users can consume GPUs across different Kubernetes versions and the current limitations.

## Using device plugins

Kubernetes implements [Device Plugins](#) to let Pods access specialized hardware features such as GPUs.

As an administrator, you have to install GPU drivers from the corresponding hardware vendor on the nodes and run the corresponding device plugin from the GPU vendor:

- [AMD](#)
- [NVIDIA](#)

When the above conditions are true, Kubernetes will expose `amd.com/gpu` or `nvidia.com/gpu` as a schedulable resource.

You can consume these GPUs from your containers by requesting `<vendor>.com/gpu` just like you request `cpu` or `memory`. However, there are some limitations in how you specify the resource requirements when using GPUs:

- GPUs are only supposed to be specified in the `limits` section, which means:
  - You can specify GPU limits without specifying requests because Kubernetes will use the limit as the request value by default.
  - You can specify GPU in both `limits` and `requests` but these two values must be equal.
  - You cannot specify GPU requests without specifying limits.
- Containers (and Pods) do not share GPUs. There's no overcommitting of GPUs.
- Each container can request one or more GPUs. It is not possible to request a fraction of a GPU.

Here's an example:

```
apiVersion: v1
kind: Pod
metadata:
  name: cuda-vector-add
spec:
  restartPolicy: OnFailure
  containers:
    - name: cuda-vector-add
      # https://github.com/kubernetes/kubernetes/blob/v1.7.11/test/images/nvidia-cuda/Dockerfile
      image: "k8s.gcr.io/cuda-vector-add:v0.1"
      resources:
        limits:
          nvidia.com/gpu: 1 # requesting 1 GPU
```

## Deploying AMD GPU device plugin

The [official AMD GPU device plugin](#) has the following requirements:

- Kubernetes nodes have to be pre-installed with AMD GPU Linux driver.

To deploy the AMD device plugin once your cluster is running and the above requirements are satisfied:

```
kubectl create -f https://raw.githubusercontent.com/RadeonOpenCompute/k8s-device-plugin/v1.10/k8s-ds-amdgpu-dp.yaml
```

You can report issues with this third-party device plugin by logging an issue in [RadeonOpenCompute/k8s-device-plugin](#).

## Deploying NVIDIA GPU device plugin

There are currently two device plugin implementations for NVIDIA GPUs:

### Official NVIDIA GPU device plugin

The [official NVIDIA GPU device plugin](#) has the following requirements:

- Kubernetes nodes have to be pre-installed with NVIDIA drivers.
- Kubernetes nodes have to be pre-installed with [nvidia-docker 2.0](#)
- Kubelet must use Docker as its container runtime
- `nvidia-container-runtime` must be configured as the [default runtime](#) for Docker, instead of runc.
- The version of the NVIDIA drivers must match the constraint `~> 384.81`.

To deploy the NVIDIA device plugin once your cluster is running and the above requirements are satisfied:

```
kubectl create -f https://raw.githubusercontent.com/NVIDIA/k8s-device-plugin/1.0.0-beta4/nvidia-device-plugin.yaml
```

You can report issues with this third-party device plugin by logging an issue in [NVIDIA/k8s-device-plugin](#).

### NVIDIA GPU device plugin used by GCE

The [NVIDIA GPU device plugin used by GCE](#) doesn't require using nvidia-docker and should work with any container runtime that is compatible with the Kubernetes Container Runtime Interface (CRI). It's tested on [Container-Optimized OS](#) and has experimental code for Ubuntu from 1.9 onwards.

You can use the following commands to install the NVIDIA drivers and device plugin:

```
# Install NVIDIA drivers on Container-Optimized OS:  
kubectl create -f https://raw.githubusercontent.com/  
GoogleCloudPlatform/container-engine-accelerators/stable/  
daemonset.yaml  
  
# Install NVIDIA drivers on Ubuntu (experimental):  
kubectl create -f https://raw.githubusercontent.com/  
GoogleCloudPlatform/container-engine-accelerators/stable/  
nvidia-driver-installer/ubuntu/daemonset.yaml  
  
# Install the device plugin:  
kubectl create -f https://raw.githubusercontent.com/  
kubernetes/kubernetes/release-1.14/cluster/addons/device-  
plugins/nvidia-gpu/daemonset.yaml
```

You can report issues with using or deploying this third-party device plugin by logging an issue in [GoogleCloudPlatform/container-engine-accelerators](#).

Google publishes its own [instructions](#) for using NVIDIA GPUs on GKE .

## Clusters containing different types of GPUs

If different nodes in your cluster have different types of GPUs, then you can use [Node Labels and Node Selectors](#) to schedule pods to appropriate nodes.

For example:

```
# Label your nodes with the accelerator type they have.  
kubectl label nodes <node-with-k80> accelerator=nvidia-tesla-  
k80
```

```
kubectl label nodes <node-with-p100> accelerator=nvidia-tesla-p100
```

## Automatic node labelling

If you're using AMD GPU devices, you can deploy [Node Labeler](#). Node Labeler is a [controller](#) that automatically labels your nodes with GPU device properties.

At the moment, that controller can add labels for:

- Device ID (-device-id)
- VRAM Size (-vram)
- Number of SIMD (-simd-count)
- Number of Compute Unit (-cu-count)
- Firmware and Feature Versions (-firmware)
- GPU Family, in two letters acronym (-family)
  - SI - Southern Islands
  - CI - Sea Islands
  - KV - Kaveri
  - VI - Volcanic Islands
  - CZ - Carrizo
  - AI - Arctic Islands
  - RV - Raven

```
kubectl describe node cluster-node-23
```

Name:	cluster-node-23
Roles:	<none>
Labels:	beta.amd.com/gpu.cu-count.64=1 beta.amd.com/gpu.device-id.6860=1 beta.amd.com/gpu.family.AI=1 beta.amd.com/gpu.simd-count.256=1 beta.amd.com/gpu.vram.16G=1 beta.kubernetes.io/arch=amd64 beta.kubernetes.io/os=linux kubernetes.io/hostname=cluster-node-23
Annotations:	kubeadm.alpha.kubernetes.io/cri-socket: /var/run/dockershim.sock node.alpha.kubernetes.io/ttl: 0
	â€¢

With the Node Labeler in use, you can specify the GPU type in the Pod spec:

```
apiVersion: v1
kind: Pod
metadata:
  name: cuda-vector-add
spec:
  restartPolicy: OnFailure
```

```
  containers:
    - name: cuda-vector-add
      # https://github.com/kubernetes/kubernetes/blob/v1.7.11/test/images/nvidia-cuda/Dockerfile
      image: "k8s.gcr.io/cuda-vector-add:v0.1"
      resources:
        limits:
          nvidia.com/gpu: 1
      nodeSelector:
        accelerator: nvidia-tesla-p100 # or nvidia-tesla-k80 etc.
```

This will ensure that the Pod will be scheduled to a node that has the GPU type you 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](#).

Last modified June 14, 2020 at 8:35 PM PST: [Add descriptions for tasks \(74360fa8e\)](#)

[Edit this page](#) [Create child page](#) [Create an issue](#)

- [Using device plugins](#)
  - [Deploying AMD GPU device plugin](#)
  - [Deploying NVIDIA GPU device plugin](#)
- [Clusters containing different types of GPUs](#)
- [Automatic node labelling](#)