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**](https://kubernetes.io/docs/tasks/access-application-cluster/configure-access-multiple-clusters/#before-you-begin)
* [**Define clusters, users, and contexts**](https://kubernetes.io/docs/tasks/access-application-cluster/configure-access-multiple-clusters/#define-clusters-users-and-contexts)
* [**Create a second configuration file**](https://kubernetes.io/docs/tasks/access-application-cluster/configure-access-multiple-clusters/#create-a-second-configuration-file)
* [**Set the KUBECONFIG environment variable**](https://kubernetes.io/docs/tasks/access-application-cluster/configure-access-multiple-clusters/#set-the-kubeconfig-environment-variable)
* [**Explore the $HOME/.kube directory**](https://kubernetes.io/docs/tasks/access-application-cluster/configure-access-multiple-clusters/#explore-the-home-kube-directory)
* [**Append $HOME/.kube/config to your KUBECONFIG environment variable**](https://kubernetes.io/docs/tasks/access-application-cluster/configure-access-multiple-clusters/#append-home-kube-config-to-your-kubeconfig-environment-variable)
* [**Clean up**](https://kubernetes.io/docs/tasks/access-application-cluster/configure-access-multiple-clusters/#clean-up)
* [**What's next**](https://kubernetes.io/docs/tasks/access-application-cluster/configure-access-multiple-clusters/#what-s-next)

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](https://kubernetes.io/docs/setup/minikube), or you can use one of these Kubernetes playgrounds:

* [Katacoda](https://www.katacoda.com/courses/kubernetes/playground)
* [Play with Kubernetes](http://labs.play-with-k8s.com/)

To check the version, enter kubectl version.

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 config unset users.<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 is the placeholders for the real path of the certification files. You need change these to the real path of certification files in your environment.

Some times you may want to use base64 encoded data here instead of the path of the certification files, then 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](https://kubernetes.io/docs/concepts/configuration/organize-cluster-access-kubeconfig/)

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 KUBECONFIG 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