# Access kubectl using GCP SDK

https://cloud.google.com/kubernetes-engine/docs/how-to/cluster-access-for-kubectl#generate\_kubeconfig\_entry

# Creating a kubeconfig file for a self-hosted Kubernetes cluster

This tutorial explains how to create a kubeconfig file to authenticate to a self hosted Kubernetes cluster. If you use a hosted solution like GKE or AKS, you get the benefit of the cloud-providers Auth system. If it is self hosted, then it lacks this luxury. This guide helps you to create a service account on Kubernetes and create a kubeconfig file that can be used by kubectl to interact with the cluster.

It assumes that you have working knowledge of Docker and Kubernetes and understand the following concepts:

* [Self hosting on GCP](https://devopscube.com/setup-kubernetes-cluster-google-cloud/)
* [Self hosting on AWS](https://kubernetes.io/docs/setup/turnkey/aws/)
* [kubeconfig files](https://kubernetes.io/docs/concepts/configuration/organize-cluster-access-kubeconfig/)
* [Configuring Service Accounts](https://kubernetes.io/docs/tasks/configure-pod-container/configure-service-account/)

## Step by Step instructions

The following steps need to be followed carefully in order to create a service account kubeconfig file.

### Make sure you can access the cluster

First, make sure you can authenticate yourself to the cluster. This means you have a kubeconfig file that uses your personal account. You can verify this by running this command on your local machine and you should see the file listed

ls -al $HOME/.kube

### Author a service account spec

To create a service account on Kubernetes, you can leverage kubectl and a service account spec. Create a YML file name *sa.yml* that looks like the one below:

**sa.yml**

apiVersion: v1

kind: ServiceAccount

metadata:

name: svcs-acct-dply *#any name you'd like*

### Create the service account

You can create a service account by running the following command:

kubectl create -f sa.yaml

This will use your personal account to create the service account. Make sure your personal account has permissions to do this.

### Fetch the name of the secrets used by the service account

This can be found by running the following command:

kubectl describe serviceAccounts svcs-acct-dply

**output**

Name: svcs-acct-dply

Namespace: default

Labels: <none>

Annotations: <none>

Image pull secrets: <none>

Mountable secrets: svcs-acct-dply-token-h6pdj

Tokens: svcs-acct-dply-token-h6pdj

Note down the Mountable secrets information which has the name of the secret that holds the token

### Fetch the token from the secret

Using the Mountable secrets value, you can get the token used by the service account. Run the following command to extract this information:

kubectl describe secrets svcs-acct-dply-token-h6pdj

**output**

Name: svcs-acct-dply-token-h6pdj

Namespace: default

Labels: <none>

Annotations: kubernetes.io/service-account.name=svcs-acct-dply

kubernetes.io/service-account.uid=c2117d8e-3c2d-11e8-9ccd-42010a8a012f

Type: kubernetes.io/service-account-token

Data

====

ca.crt: 1115 bytes

namespace: 7 bytes

token: eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9..ZWKrKdpK7aukTRKnB5SJwwov6PjaADT-FqSO9ZgJEg6uUVXuPa03jmqyRB20HmsTvuDabVoK7Ky7Uug7V8J9yK4oOOK5d0aRRdgHXzxZd2yO8C4ggqsr1KQsfdlU4xRWglaZGI4S31ohCApJ0MUHaVnP5WkbC4FiTZAQ5fO\_LcCokapzCLQyIuD5Ksdnj5Ad2ymiLQQ71TUNccN7BMX5aM4RHmztpEHOVbElCWXwyhWr3NR1Z1ar9s5ec6iHBqfkp\_s8TvxPBLyUdy9OjCWy3iLQ4Lt4qpxsjwE4NE7KioDPX2Snb6NWFK7lvldjYX4tdkpWdQHBNmqaD8CuVCRdEQ

This will output the token information that looks something like above. Note down the token value

### Get the certificate info for the cluster

Every cluster has a certificate that clients can use to encryt traffic. Fetch the certificate and write to a file by running this command. In this case, we are using a file name cluster-cert.txt

kubectl config view --flatten --minify > cluster-cert.txt

cat cluster-cert.txt

**output**

apiVersion: v1

clusters:

- cluster:

certificate-authority-data: 

server: https://35.203.181.169

name: gke\_jfrog-200320\_us-west1-a\_cluster

contexts:

- context:

cluster: gke\_jfrog-200320\_us-west1-a\_cluster

user: gke\_jfrog-200320\_us-west1-a\_cluster

name: gke\_jfrog-200320\_us-west1-a\_cluster

current-context: gke\_jfrog-200320\_us-west1-a\_cluster

kind: Config

preferences: {}

users:

- name: gke\_jfrog-200320\_us-west1-a\_cluster

user:

auth-provider:

config:

access-token: ya29.Gl2YBba5duRR8Zb6DekAdjPtPGepx9Em3gX1LAhJuYzq1G4XpYwXTS\_wF4cieZ8qztMhB35lFJC-DJR6xcB02oXXkiZvWk5hH4YAw1FPrfsZWG57x43xCrl6cvHAp40

cmd-args: config config-helper --format=json

cmd-path: /Users/ambarish/google-cloud-sdk/bin/gcloud

expiry: 2018-04-09T20:35:02Z

expiry-key: '{.credential.token\_expiry}'

token-key: '{.credential.access\_token}'

name: gcp

Copy two pieces of information from here certificate-authority-data and server

### Create a kubeconfig file

From the steps above, you should have the following pieces of information

* token
* certificate-authority-data
* server

Create a file called sa-config and paste this content on to it

apiVersion: v1

kind: Config

users:

- name: svcs-acct-dply

user:

token: <replace this with token info>

clusters:

- cluster:

certificate-authority-data: <replace this with certificate-authority-data info>

server: <replace this with server info>

name: self-hosted-cluster

contexts:

- context:

cluster: self-hosted-cluster

user: svcs-acct-dply

name: svcs-acct-context

current-context: svcs-acct-context

Replace the placeholder above with the information gathered so far

* replace the token
* replace the certificate-authority-data
* replace the server

### Copy the file to $HOME/.kube

If you want your client to use this context, copy sa-config to $HOME/.kube and you can configure kubectl to use the context

kubectl config --kubeconfig=$HOME/.kube/sa-config set-context svcs-acct-context