Install and Set Up kubectl on macOS

Before you begin

You must use a kubectl version that is within one minor version difference of your cluster. For example, a v1.34 client can communicate with v1.33, v1.34, and v1.35 control planes. Using the latest compatible version of kubectl helps avoid unforeseen issues.

Install kubectl on macOS

The following methods exist for installing kubectl on macOS:

- Install kubectl on macOS
  - Install kubectl binary with curl on macOS
  - Install with Homebrew on macOS
  - Install with Macports on macOS
- Verify kubectl configuration
- Optional kubectl configurations and plugins
  - Enable shell autocompletion
  - Install `kubectl convert` plugin

Install kubectl binary with curl on macOS

1. Download the latest release:

   - Intel
   - Apple Silicon

   ```
   curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/darwin/amd64/kubectl"
   ```

   ```
   curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/darwin/arm64/kubectl"
   ```

   Note:

   To download a specific version, replace the `$(curl -L -s https://dl.k8s.io/release/stable.txt)` portion of the command with the specific version.

   For example, to download version 1.34.0 on Intel macOS, type:

   ```
   curl -LO "https://dl.k8s.io/release/v1.34.0/bin/darwin/amd64/kubectl"
   ```

   And for macOS on Apple Silicon, type:

   ```
   curl -LO "https://dl.k8s.io/release/v1.34.0/bin/darwin/arm64/kubectl"
   ```

2. Validate the binary (optional)

   Download the kubectl checksum file:

   - Intel
   - Apple Silicon

   ```
   curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/darwin/amd64/kubectl.sha256"
   ```

   ```
   curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/darwin/arm64/kubectl.sha256"
   ```

   Validate the kubectl binary against the checksum file:

   ```
   echo "$(cat kubectl.sha256)  kubectl" | shasum -a 256 --check
   ```

   If valid, the output is:

   ```
   kubectl: OK
   ```

   If the check fails, `shasum` exits with nonzero status and prints output similar to:

   ```
   kubectl: FAILED
   shasum: WARNING: 1 computed checksum did NOT match
   ```

   Note:

   Download the same version of the binary and checksum.

3. Make the kubectl binary executable.

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

4. Move the kubectl binary to a file location on your system `PATH`.

   ```
   sudo mv ./kubectl /usr/local/bin/kubectl
   sudo chown root: /usr/local/bin/kubectl
   ```

   Note:

   Make sure `/usr/local/bin` is in your PATH environment variable.

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

   ```
   kubectl version --client
   ```

   Or use this for detailed view of version:

   ```
   kubectl version --client --output=yaml
   ```

6. After installing and validating kubectl, delete the checksum file:

   ```
   rm kubectl.sha256
   ```

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
   ```

Verify 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, check whether it is configured properly using the following command:

```
kubectl cluster-info dump
```

Troubleshooting the 'No Auth Provider Found' error message

In Kubernetes 1.26, kubectl removed the built-in authentication for the following cloud providers' managed Kubernetes offerings. These providers have released kubectl plugins to provide the cloud-specific authentication. For instructions, refer to the following provider documentation:

- Azure AKS: kubelogin plugin
- Google Kubernetes Engine: gke-gcloud-auth-plugin

There could also be other causes for the same error message that are unrelated to that change.

Optional kubectl configurations and plugins

Enable shell autocompletion

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

Below are the procedures to set up autocompletion for Bash, Fish, and Zsh.

- Bash
- Fish
- Zsh

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:

```
brew_etc="$(brew --prefix)/etc" && [[ -r "${brew_etc}/profile.d/bash_completion.sh" ]] && . "${brew_etc}/profile.d/bash_completion
```

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 -o default -F __start_kubectl k' >>~/.bash_profile
  ```

- If you installed kubectl with Homebrew (as explained here), 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.

Note:

Autocomplete for Fish requires kubectl 1.23 or later.

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

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

```
kubectl completion fish | source
```

After reloading your shell, kubectl autocompletion 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, kubectl autocompletion will automatically work with it.

After reloading your shell, kubectl autocompletion should be working.

If you get an error like `2: command not found: compdef`, then add the following to the beginning of your `~/.zshrc` file:

```
autoload -Uz compinit
compinit
```

Configure kuberc

See kuberc for more information.

Install `kubectl convert` plugin

A plugin for Kubernetes command-line tool `kubectl`, which allows you to convert manifests between different API versions. This can be particularly helpful to migrate manifests to a non-deprecated api version with newer Kubernetes release. For more info, visit migrate to non deprecated apis

1. Download the latest release with the command:

   - Intel
   - Apple Silicon

   ```
   curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/darwin/amd64/kubectl-convert"
   ```

   ```
   curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/darwin/arm64/kubectl-convert"
   ```

2. Validate the binary (optional)

   Download the kubectl-convert checksum file:

   - Intel
   - Apple Silicon

   ```
   curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/darwin/amd64/kubectl-convert.sh
   ```

   ```
   curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/darwin/arm64/kubectl-convert.sh
   ```

   Validate the kubectl-convert binary against the checksum file:

   ```
   echo "$(cat kubectl-convert.sha256)  kubectl-convert" | shasum -a 256 --check
   ```

   If valid, the output is:

   ```
   kubectl-convert: OK
   ```

   If the check fails, `shasum` exits with nonzero status and prints output similar to:

   ```
   kubectl-convert: FAILED
   shasum: WARNING: 1 computed checksum did NOT match
   ```

   Note:

Download the same version of the binary and checksum.

3. Make kubectl-convert binary executable

   ```
   chmod +x ./kubectl-convert
   ```

4. Move the kubectl-convert binary to a file location on your system `PATH`.

   ```
   sudo mv ./kubectl-convert /usr/local/bin/kubectl-convert
   sudo chown root: /usr/local/bin/kubectl-convert
   ```

   Note:

   Make sure `/usr/local/bin` is in your PATH environment variable.

5. Verify plugin is successfully installed

   ```
   kubectl convert --help
   ```

   If you do not see an error, it means the plugin is successfully installed.

6. After installing the plugin, clean up the installation files:

   ```
   rm kubectl-convert kubectl-convert.sha256
   ```

Uninstall kubectl on macOS

Depending on how you installed `kubectl`, use one of the following methods.

Uninstall kubectl using the command-line

1. Locate the `kubectl` binary on your system:

   ```
   which kubectl
   ```

2. Remove the `kubectl` binary:

   ```
   sudo rm <path>
   ```

   Replace `<path>` with the path to the `kubectl` binary from the previous step. For example, `sudo rm /usr/local/bin/kubectl`.

Uninstall kubectl using homebrew

If you installed `kubectl` using Homebrew, run the following command:

```
brew remove kubectl
```

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

Issue a Certificate for a Kubernetes API Client Using A CertificateSigningRequest

Kubernetes lets you use a public key infrastructure (PKI) to authenticate to your cluster as a client.

A few steps are required in order to get a normal user to be able to authenticate and invoke an API. First, this user must have an X.509 certificate issued by an authority that your Kubernetes cluster trusts. The client must then present that certificate to the Kubernetes API.

You use a CertificateSigningRequest as part of this process, and either you or some other principal must approve the request.

You will create a private key, and then get a certificate issued, and finally configure that private key for a client.

Before you begin

- You need to have a Kubernetes cluster, and the kubectl command-line tool must be configured to communicate with your cluster. It is recommended to run this tutorial on a cluster with at least two nodes that are not acting as control plane hosts. If you do not already have a cluster, you can create one by using minikube or you can use one of these Kubernetes playgrounds:
  - iximiuz Labs
  - Killercoda
  - KodeKloud
  - Play with Kubernetes

- You need the `kubectl`, `openssl` and `base64` utilities.

This page assumes you are using Kubernetes role based access control (RBAC). If you have alternative or additional security mechanisms around authorization, you need to account for those as well.

Create private key

In this step, you create a private key. You need to keep this document secret; anyone who has it can impersonate the user.

```
# Create a private key
openssl genrsa -out myuser.key 3072
```

Create an X.509 certificate signing request

Note:

This is not the same as the similarly-named CertificateSigningRequest API; the file you generate here goes into the CertificateSigningRequest.

It is important to set CN and O attribute of the CSR. CN is the name of the user and O is the group that this user will belong to. You can refer to RBAC for standard groups.

```
# Change the common name "myuser" to the actual username that you want to use
openssl req -new -key myuser.key -out myuser.csr -subj "/CN=myuser"
```

Create a Kubernetes CertificateSigningRequest

Encode the CSR document using this command:

```
cat myuser.csr | base64 | tr -d "\n"
```

Create a CertificateSigningRequest and submit it to a Kubernetes Cluster via kubectl. Below is a snippet of shell that you can use to generate the CertificateSigningRequest.

```
cat <<EOF | kubectl apply -f -
apiVersion: certificates.k8s.io/v1
kind: CertificateSigningRequest
metadata:
  name: myuser # example
spec:
  # This is an encoded CSR. Change this to the base64-encoded contents of myuser.csr
  request: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURSBSRVFVRVNULS0tLS0KTUlJQ1ZqQ0NBVDRDQVFBd0VURVBNQTBHQTFVRUF3d0dZVzVuWld4aElJSUJJakFOQmdrcWh
  signerName: kubernetes.io/kube-apiserver-client
  expirationSeconds: 86400  # one day
  usages:
  - client auth
EOF
```

Some points to note:

- `usages` has to be `client auth`
- `expirationSeconds` could be made longer (i.e. `864000` for ten days) or shorter (i.e. `3600` for one hour). You cannot request a duration shorter than 10 minutes.
- `request` is the base64 encoded value of the CSR file content.

Approve the CertificateSigningRequest

Use kubectl to find the CSR you made, and manually approve it.

Get the list of CSRs:

```
kubectl get csr
```

Approve the CSR:

```
kubectl certificate approve myuser
```

Get the certificate

Retrieve the certificate from the CSR, to check it looks OK.

```
kubectl get csr/myuser -o yaml
```

The certificate value is in Base64-encoded format under `.status.certificate`.

Export the issued certificate from the CertificateSigningRequest.

```
kubectl get csr myuser -o jsonpath='{.status.certificate}'| base64 -d > myuser.crt
```

Configure the certificate into kubeconfig

The next step is to add this user into the kubeconfig file.

First, you need to add new credentials:

```
kubectl config set-credentials myuser --client-key=myuser.key --client-certificate=myuser.crt --embed-certs=true
```

Then, you need to add the context:

```
kubectl config set-context myuser --cluster=kubernetes --user=myuser
```

To test it:

```
kubectl --context myuser auth whoami
```

You should see output confirming that you are “myuser“.

Create Role and RoleBinding

Note:

If you don't use Kubernetes RBAC, skip this step and make the appropriate changes for the authorization mechanism your cluster actually uses.

With the certificate created it is time to define the Role and RoleBinding for this user to access Kubernetes cluster resources.

This is a sample command to create a Role for this new user:

```
kubectl create role developer --verb=create --verb=get --verb=list --verb=update --verb=delete --resource=pods
```

This is a sample command to create a RoleBinding for this new user:

```
kubectl create rolebinding developer-binding-myuser --role=developer --user=myuser
```

What's next

- Read Manage TLS Certificates in a Cluster
- For details of X.509 itself, refer to RFC 5280 section 3.1
- For information on the syntax of PKCS#10 certificate signing requests, refer to RFC 2986
- Read about ClusterTrustBundles

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. It is recommended to run this tutorial on a cluster with at least two nodes that are not acting as control plane hosts. If you do not already have a cluster, you can create one by using minikube or you can use one of these Kubernetes playgrounds:

- iximiuz Labs
- Killercoda
- KodeKloud
- Play with Kubernetes

- 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 (for example: `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 the `--root-ca-file` flag for the kube-controller-manager to include both old and new CA, then restart the kube-controller-manager.

   Any ServiceAccount 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.

Issue 1350 for kubeadm tracks an bug with the kube-controller-manager being unable to accept a CA bundle.

3. Wait for the controller manager to update `ca.crt` in the service account Secrets to include both old and new CA certificates.

   If any Pods are started before new CA is used by API servers, the new Pods get this update and will trust both old and new CAs.

4. Restart all pods using in-cluster configurations (for example: kube-proxy, CoreDNS, etc) so they can use the updated certificate authority data from Secrets that link to ServiceAccounts.

   - Make sure CoreDNS, kube-proxy and other Pods using in-cluster configurations 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. Update the `--root-ca-file` flag for the Cloud Controller Manager to include both old and new CA, then restart the cloud-controller-manager.

   Note:

   If your cluster does not have a cloud-controller-manager, you can skip this step.

9. Follow the steps below 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`, `apiserver-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 Pods in your cluster trust both old and new CAs, there will be a momentarily disconnection after which pods' Kubernetes clients reconnect to the new API server. The new API server uses a certificate signed by the new CA.

      - Restart the kube-scheduler to use and trust 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`)](/docs/tasks/administer-cluster/certificates/#openssl).
      You can also use [`cfssl`](/docs/tasks/administer-cluster/certificates/#cfssl).
      ```

   4. Annotate any DaemonSets and Deployments to trigger pod replacement in a safer rolling fashion.

```
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": "
    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](/docs/tasks/run-application/configure-pdb/).
   ```

   ```
   Depending on how you use StatefulSets you may also need to perform similar rolling replacement.
   ```

10. 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 -w0 /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 -
```

11. Verify the cluster functionality.

1. Check the logs from control plane components, along with the kubelet and the kube-proxy. Ensure those components are not reporting any TLS errors; see looking at the logs for more details.

2. Validate logs from any aggregated api servers and pods using in-cluster config.

12. Once the cluster functionality is successfully verified:

    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 Secret, so that no Pods are relying on the old cluster CA.

    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. On each node, restart the kubelet by removing the old CA from file against the `clientCAFile` flag and from the kubelet kubeconfig file. You should carry this out as a rolling update.

       If your cluster lets you make this change, you can also roll it out by replacing nodes rather than reconfiguring them.

Install and Set Up kubectl on Windows

Before you begin

You must use a kubectl version that is within one minor version difference of your cluster. For example, a v1.34 client can communicate with v1.33, v1.34, and v1.35 control planes. Using the latest compatible version of kubectl helps avoid unforeseen issues.

Install kubectl on Windows

The following methods exist for installing kubectl on Windows:

- Install kubectl binary on Windows (via direct download or curl)
- Install on Windows using Chocolatey, Scoop, or winget

Install kubectl binary on Windows (via direct download or curl)

1. You have two options for installing kubectl on your Windows device

   - Direct download:

     Download the latest 1.34 patch release binary directly for your specific architecture by visiting the Kubernetes release page. Be sure to select the correct binary for your architecture (e.g., amd64, arm64, etc.).

   - Using curl:

     If you have `curl` installed, use this command:

     ```
     curl.exe -LO "https://dl.k8s.io/release/v1.34.0/bin/windows/amd64/kubectl.exe"
     ```

   Note:

   To find out the latest stable version (for example, for scripting), take a look at https://dl.k8s.io/release/stable.txt.

2. Validate the binary (optional)

   Download the `kubectl` checksum file:

   ```
   curl.exe -LO "https://dl.k8s.io/v1.34.0/bin/windows/amd64/kubectl.exe.sha256"
   ```

   Validate the `kubectl` binary against the checksum file:

   - Using Command Prompt to manually compare `CertUtil`'s output to the checksum file downloaded:

     ```
     CertUtil -hashfile kubectl.exe SHA256
     type kubectl.exe.sha256
     ```

   - Using PowerShell to automate the verification using the `-eq` operator to get a `True` or `False` result:

     ```
     $(Get-FileHash -Algorithm SHA256 .\kubectl.exe).Hash -eq $(Get-Content .\kubectl.exe.sha256)
     ```

3. Append or prepend the `kubectl` binary folder to your `PATH` environment variable.

4. Test to ensure the version of `kubectl` is the same as downloaded:

   ```
   kubectl version --client
   ```

   Or use this for detailed view of version:

   ```
   kubectl version --client --output=yaml
   ```

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 on Windows using Chocolatey, Scoop, or winget

1. To install kubectl on Windows you can use either Chocolatey package manager, Scoop command-line installer, or winget package manager.

   - choco
   - scoop
   - winget

   ```
   choco install kubernetes-cli
   ```

   ```
   scoop install kubectl
   ```

   ```
   winget install -e --id Kubernetes.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.

Verify 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, check whether it is configured properly using the following command:

```
kubectl cluster-info dump
```

Troubleshooting the 'No Auth Provider Found' error message

In Kubernetes 1.26, kubectl removed the built-in authentication for the following cloud providers' managed Kubernetes offerings. These providers have released kubectl plugins to provide the cloud-specific authentication. For instructions, refer to the following provider documentation:

- Azure AKS: kubelogin plugin
- Google Kubernetes Engine: gke-gcloud-auth-plugin

There could also be other causes for the same error message that are unrelated to that change.

Optional kubectl configurations and plugins

Enable shell autocompletion

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

Below are the procedures to set up autocompletion for PowerShell.

The kubectl completion script for PowerShell can be generated with the command `kubectl completion powershell`.

To do so in all your shell sessions, add the following line to your `$PROFILE` file:

```
kubectl completion powershell | Out-String | Invoke-Expression
```

This command will regenerate the auto-completion script on every PowerShell start up. You can also add the generated script directly to your `$PROFILE` file.

To add the generated script to your `$PROFILE` file, run the following line in your powershell prompt:

```
kubectl completion powershell >> $PROFILE
```

After reloading your shell, kubectl autocompletion should be working.

Configure kuberc

See kuberc for more information.

Install `kubectl convert` plugin

A plugin for Kubernetes command-line tool `kubectl`, which allows you to convert manifests between different API versions. This can be particularly helpful to migrate manifests to a non-deprecated api version with newer Kubernetes release. For more info, visit migrate to non deprecated apis

1. Download the latest release with the command:

   ```
   curl.exe -LO "https://dl.k8s.io/release/v1.34.0/bin/windows/amd64/kubectl-convert.exe"
   ```

2. Validate the binary (optional).

   Download the `kubectl-convert` checksum file:

   ```
   curl.exe -LO "https://dl.k8s.io/v1.34.0/bin/windows/amd64/kubectl-convert.exe.sha256"
   ```

   Validate the `kubectl-convert` binary against the checksum file:

   - Using Command Prompt to manually compare `CertUtil`'s output to the checksum file downloaded:

     ```
     CertUtil -hashfile kubectl-convert.exe SHA256
     type kubectl-convert.exe.sha256
     ```

   - Using PowerShell to automate the verification using the `-eq` operator to get a `True` or `False` result:

     ```
     $($(CertUtil -hashfile .\kubectl-convert.exe SHA256)[1] -replace " ", "") -eq $(type .\kubectl-convert.exe.sha256)
     ```

3. Append or prepend the `kubectl-convert` binary folder to your `PATH` environment variable.

4. Verify the plugin is successfully installed.

   ```
   kubectl convert --help
   ```

   If you do not see an error, it means the plugin is successfully installed.

5. After installing the plugin, clean up the installation files:

   ```
   del kubectl-convert.exe
   del kubectl-convert.exe.sha256
   ```

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

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 contains kubelet certificate rotation, 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 request.

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. This can happen at any point between 30% and 10% of the time remaining on the certificate. 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.

Install and Set Up kubectl on Linux

Before you begin

You must use a kubectl version that is within one minor version difference of your cluster. For example, a v1.34 client can communicate with v1.33, v1.34, and v1.35 control planes. Using the latest compatible version of kubectl helps avoid unforeseen issues.

Install kubectl on Linux

The following methods exist for installing kubectl on Linux:

- Install kubectl binary with curl on Linux
- Install using native package management
- Install using other package management

Install kubectl binary with curl on Linux

1. Download the latest release with the command:

   - x86-64
   - ARM64

   ```
   curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"
   ```

   ```
   curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/arm64/kubectl"
   ```

   Note:

   To download a specific version, replace the `$(curl -L -s https://dl.k8s.io/release/stable.txt)` portion of the command with the specific version.

   For example, to download version 1.34.0 on Linux x86-64, type:

   ```
   curl -LO https://dl.k8s.io/release/v1.34.0/bin/linux/amd64/kubectl
   ```

   And for Linux ARM64, type:

   ```
   curl -LO https://dl.k8s.io/release/v1.34.0/bin/linux/arm64/kubectl
   ```

2. Validate the binary (optional)

   Download the kubectl checksum file:

   - x86-64
   - ARM64

   ```
   curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl.sha256"
   ```

```
curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/arm64/kubectl.sha256"
```

Validate the kubectl binary against the checksum file:

```
echo "$(cat kubectl.sha256)  kubectl" | sha256sum --check
```

If valid, the output is:

```
kubectl: OK
```

If the check fails, `sha256` exits with nonzero status and prints output similar to:

```
kubectl: FAILED
sha256sum: WARNING: 1 computed checksum did NOT match
```

Note:

Download the same version of the binary and checksum.

3. Install kubectl

```
sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl
```

Note:

If you do not have root access on the target system, you can still install kubectl to the `~/.local/bin` directory:

```
chmod +x kubectl
mkdir -p ~/.local/bin
mv ./kubectl ~/.local/bin/kubectl
# and then append (or prepend) ~/.local/bin to $PATH
```

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

```
kubectl version --client
```

Or use this for detailed view of version:

```
kubectl version --client --output=yaml
```

Install using native package management

- Debian-based distributions
- Red Hat-based distributions
- SUSE-based distributions

1. Update the `apt` package index and install packages needed to use the Kubernetes `apt` repository:

```
sudo apt-get update
# apt-transport-https may be a dummy package; if so, you can skip that package
sudo apt-get install -y apt-transport-https ca-certificates curl gnupg
```

2. Download the public signing key for the Kubernetes package repositories. The same signing key is used for all repositories so you can disregard the version in the URL:

```
# If the folder `/etc/apt/keyrings` does not exist, it should be created before the curl command, read the note below.
# sudo mkdir -p -m 755 /etc/apt/keyrings
curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.34/deb/Release.key | sudo gpg --dearmor -o /etc/apt/keyrings/kubernetes-apt-k
sudo chmod 644 /etc/apt/keyrings/kubernetes-apt-keyring.gpg # allow unprivileged APT programs to read this keyring
```

Note:

In releases older than Debian 12 and Ubuntu 22.04, folder `/etc/apt/keyrings` does not exist by default, and it should be created before the curl command.

3. Add the appropriate Kubernetes `apt` repository. If you want to use Kubernetes version different than v1.34, replace v1.34 with the desired minor version in the command below:

```
# This overwrites any existing configuration in /etc/apt/sources.list.d/kubernetes.list
echo 'deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg] https://pkgs.k8s.io/core:/stable:/v1.34/deb/ /' | sudo tee
sudo chmod 644 /etc/apt/sources.list.d/kubernetes.list   # helps tools such as command-not-found to work correctly
```

Note:

To upgrade kubectl to another minor release, you'll need to bump the version in `/etc/apt/sources.list.d/kubernetes.list` before running `apt-get update` and `apt-get upgrade`. This procedure is described in more detail in Changing The Kubernetes Package Repository.

4. Update `apt` package index, then install kubectl:

```
sudo apt-get update
sudo apt-get install -y kubectl
```

1. Add the Kubernetes `yum` repository. If you want to use Kubernetes version different than v1.34, replace v1.34 with the desired minor version in the command below.

   ```
   # This overwrites any existing configuration in /etc/yum.repos.d/kubernetes.repo
   cat <<EOF | sudo tee /etc/yum.repos.d/kubernetes.repo
   [kubernetes]
   name=Kubernetes
   baseurl=https://pkgs.k8s.io/core:/stable:/v1.34/rpm/
   enabled=1
   gpgcheck=1
   gpgkey=https://pkgs.k8s.io/core:/stable:/v1.34/rpm/repodata/repomd.xml.key
   EOF
   ```

Note:

To upgrade kubectl to another minor release, you'll need to bump the version in `/etc/yum.repos.d/kubernetes.repo` before running `yum update`. This procedure is described in more detail in Changing The Kubernetes Package Repository.

2. Install kubectl using `yum`:

   ```
   sudo yum install -y kubectl
   ```

1. Add the Kubernetes `zypper` repository. If you want to use Kubernetes version different than v1.34, replace v1.34 with the desired minor version in the command below.

   ```
   # This overwrites any existing configuration in /etc/zypp/repos.d/kubernetes.repo
   cat <<EOF | sudo tee /etc/zypp/repos.d/kubernetes.repo
   [kubernetes]
   name=Kubernetes
   baseurl=https://pkgs.k8s.io/core:/stable:/v1.34/rpm/
   enabled=1
   gpgcheck=1
   gpgkey=https://pkgs.k8s.io/core:/stable:/v1.34/rpm/repodata/repomd.xml.key
   EOF
   ```

Note:

To upgrade kubectl to another minor release, you'll need to bump the version in `/etc/zypp/repos.d/kubernetes.repo` before running `zypper update`. This procedure is described in more detail in Changing The Kubernetes Package Repository.

2. Update `zypper` and confirm the new repo addition:

   ```
   sudo zypper update
   ```

   When this message appears, press 't' or 'a':

   ```
   New repository or package signing key received:

   Repository:       Kubernetes
   Key Fingerprint:  1111 2222 3333 4444 5555 6666 7777 8888 9999 AAAA
   Key Name:         isv:kubernetes OBS Project <isv:kubernetes@build.opensuse.org>
   Key Algorithm:    RSA 2048
   Key Created:      Thu 25 Aug 2022 01:21:11 PM -03
   Key Expires:      Sat 02 Nov 2024 01:21:11 PM -03 (expires in 85 days)
   Rpm Name:         gpg-pubkey-9a296436-6307a177

   Note: Signing data enables the recipient to verify that no modifications occurred after the data
   were signed. Accepting data with no, wrong or unknown signature can lead to a corrupted system
   and in extreme cases even to a system compromise.

   Note: A GPG pubkey is clearly identified by its fingerprint. Do not rely on the key's name. If
   you are not sure whether the presented key is authentic, ask the repository provider or check
   their web site. Many providers maintain a web page showing the fingerprints of the GPG keys they
   are using.

   Do you want to reject the key, trust temporarily, or trust always? [r/t/a/?] (r): a
   ```

3. Install kubectl using `zypper`:

   ```
   sudo zypper install -y kubectl
   ```

Install using other package management

- Snap
- Homebrew

If you are on Ubuntu or another Linux distribution that supports the 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
```

Verify 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, check whether it is configured properly using the following command:

```
kubectl cluster-info dump
```

Troubleshooting the 'No Auth Provider Found' error message

In Kubernetes 1.26, kubectl removed the built-in authentication for the following cloud providers' managed Kubernetes offerings. These providers have released kubectl plugins to provide the cloud-specific authentication. For instructions, refer to the following provider documentation:

- Azure AKS: kubelogin plugin
- Google Kubernetes Engine: gke-gcloud-auth-plugin

There could also be other causes for the same error message that are unrelated to that change.

Optional kubectl configurations and plugins

Enable shell autocompletion

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

Below are the procedures to set up autocompletion for Bash, Fish, and Zsh.

- Bash
- Fish
- 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 autocompletion.

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

Bash

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:

- User
- System

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

```
kubectl completion bash | sudo tee /etc/bash_completion.d/kubectl > /dev/null
sudo chmod a+r /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 -o default -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. To enable bash autocompletion in current session of shell, source the ~/.bashrc file:

```
source ~/.bashrc
```

Note:

Autocomplete for Fish requires kubectl 1.23 or later.

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

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

```
kubectl completion fish | source
```

After reloading your shell, kubectl autocompletion 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, kubectl autocompletion will automatically work with it.

After reloading your shell, kubectl autocompletion should be working.

If you get an error like `2: command not found: compdef`, then add the following to the beginning of your `~/.zshrc` file:

```
autoload -Uz compinit
compinit
```

Configure kuberc

See kuberc for more information.

Install `kubectl convert` plugin

A plugin for Kubernetes command-line tool `kubectl`, which allows you to convert manifests between different API versions. This can be particularly helpful to migrate manifests to a non-deprecated api version with newer Kubernetes release. For more info, visit migrate to non deprecated apis

1. Download the latest release with the command:

   - x86-64
   - ARM64

   ```
   curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl-convert"
   ```

   ```
   curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/arm64/kubectl-convert"
   ```

2. Validate the binary (optional)

   Download the kubectl-convert checksum file:

   - x86-64
   - ARM64

   ```
   curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl-convert.sha
   ```

   ```
   curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/arm64/kubectl-convert.sha
   ```

   Validate the kubectl-convert binary against the checksum file:

   ```
   echo "$(cat kubectl-convert.sha256) kubectl-convert" | sha256sum --check
   ```

   If valid, the output is:

   ```
   kubectl-convert: OK
   ```

If the check fails, `sha256` exits with nonzero status and prints output similar to:

```
kubectl-convert: FAILED
sha256sum: WARNING: 1 computed checksum did NOT match
```

Note:

Download the same version of the binary and checksum.

3. Install kubectl-convert

```
sudo install -o root -g root -m 0755 kubectl-convert /usr/local/bin/kubectl-convert
```

4. Verify plugin is successfully installed

```
kubectl convert --help
```

If you do not see an error, it means the plugin is successfully installed.

5. After installing the plugin, clean up the installation files:

```
rm kubectl-convert kubectl-convert.sha256
```

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

TLS

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

Issue a Certificate for a Kubernetes API Client Using A CertificateSigningRequest

Configure Certificate Rotation for the Kubelet

Manage TLS Certificates in a Cluster

Manual Rotation of CA Certificates

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. It is recommended to run this tutorial on a cluster with at least two nodes that are not acting as control plane hosts. If you do not already have a cluster, you can create one by using minikube or you can use one of these Kubernetes playgrounds:

- iximiuz Labs
- Killercoda
- KodeKloud
- Play with Kubernetes

You need the `cfssl` tool. You can download `cfssl` from https://github.com/cloudflare/cfssl/releases.

Some steps in this page use the `jq` tool. If you don't have `jq`, you can install it via your operating system's software sources, or fetch it from https://jqlang.github.io/jq/.

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.

Note:

Even though the custom CA certificate may be included in the filesystem (in the ConfigMap `kube-root-ca.crt`), you should not use that certificate authority for any purpose other than to verify internal Kubernetes endpoints. An example of an internal Kubernetes endpoint is the Service named `kubernetes` in the default namespace.

If you want to use a custom certificate authority for your workloads, you should generate that CA separately, and distribute its CA certificate using a ConfigMap that your pods have access to read.

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.

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": "my-pod.my-namespace.pod.cluster.local",
  "key": {
    "algo": "ecdsa",
    "size": 256
  }
}
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 output similar to:

```
2022/02/01 11:45:32 [INFO] generate received request
2022/02/01 11:45:32 [INFO] received CSR
2022/02/01 11:45:32 [INFO] generating key: ecdsa-256
2022/02/01 11:45:32 [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 CertificateSigningRequest object to send to the Kubernetes API

Generate a CSR manifest (in YAML), and send it to the API server. You can do that 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: example.com/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. You are also requesting a certificate with the "digital signature", "key encipherment", and "server auth" key usages, signed by an example `example.com/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:                   my-svc.my-namespace
Labels:                 <none>
Annotations:            <none>
CreationTimestamp:      Tue, 01 Feb 2022 11:49:15 -0500
Requesting User:        yourname@example.com
Signer:                 example.com/serving
Status:                 Pending
Subject:
```

```
        Common Name:    my-pod.my-namespace.pod.cluster.local
        Serial Number:
Subject Alternative Names:
        DNS Names:      my-pod.my-namespace.pod.cluster.local
                        my-svc.my-namespace.svc.cluster.local
        IP Addresses:   192.0.2.24
                        10.0.34.2
Events: <none>
```

Get the CertificateSigningRequest approved

Approving the certificate signing request is either done by an automated approval process or on a one off basis by a cluster administrator. If you're authorized to approve a certificate request, you can do that manually using `kubectl`; for example:

```
kubectl certificate approve my-svc.my-namespace
```

```
certificatesigningrequest.certificates.k8s.io/my-svc.my-namespace approved
```

You should now see the following:

```
kubectl get csr
```

```
NAME                  AGE   SIGNERNAME            REQUESTOR              REQUESTEDDURATION   CONDITION
my-svc.my-namespace   10m   example.com/serving   yourname@example.com   <none>              Approved
```

This means the certificate request has been approved and is waiting for the requested signer to sign it.

Sign the CertificateSigningRequest

Next, you'll play the part of a certificate signer, issue the certificate, and upload it to the API.

A signer would typically watch the CertificateSigningRequest API for objects with its `signerName`, check that they have been approved, sign certificates for those requests, and update the API object status with the issued certificate.

Create a Certificate Authority

You need an authority to provide the digital signature on the new certificate.

First, create a signing certificate by running the following:

```
cat <<EOF | cfssl gencert -initca - | cfssljson -bare ca
{
  "CN": "My Example Signer",
  "key": {
    "algo": "rsa",
    "size": 2048
  }
}
EOF
```

You should see output similar to:

```
2022/02/01 11:50:39 [INFO] generating a new CA key and certificate from CSR
2022/02/01 11:50:39 [INFO] generate received request
2022/02/01 11:50:39 [INFO] received CSR
2022/02/01 11:50:39 [INFO] generating key: rsa-2048
2022/02/01 11:50:39 [INFO] encoded CSR
2022/02/01 11:50:39 [INFO] signed certificate with serial number 263983151013686720899716354349605500797834580472
```

This produces a certificate authority key file (`ca-key.pem`) and certificate (`ca.pem`).

Issue a certificate

tls/server-signing-config.json 

```
{
    "signing": {
        "default": {
            "usages": [
                "digital signature",
                "key encipherment",
                "server auth"
            ],
            "expiry": "876000h",
            "ca_constraint": {
                "is_ca": false
            }
        }
    }
}
```

Use a `server-signing-config.json` signing configuration and the certificate authority key file and certificate to sign the certificate request:

```
kubectl get csr my-svc.my-namespace -o jsonpath='{.spec.request}' | \
  base64 --decode | \  cfssl sign -ca ca.pem -ca-key ca-key.pem -config server-signing-config.json - | \  cfssljson -bare ca-signe
```

You should see the output similar to:

```
2022/02/01 11:52:26 [INFO] signed certificate with serial number 576048928624926584381415936700914530534472870337
```

This produces a signed serving certificate file, `ca-signed-server.pem`.

Upload the signed certificate

Finally, populate the signed certificate in the API object's status:

```
kubectl get csr my-svc.my-namespace -o json | \
  jq '.status.certificate = "'$(base64 ca-signed-server.pem | tr -d '\n')'"' | \  kubectl replace --raw /apis/certificates.k8s.io/
```

Note:

This uses the command line tool `jq` to populate the base64-encoded content in the `.status.certificate` field. If you do not have `jq`, you can also save the JSON output to a file, populate this field manually, and upload the resulting file.

Once the CSR is approved and the signed certificate is uploaded, run:

```
kubectl get csr
```

The output is similar to:

```
NAME                  AGE   SIGNERNAME            REQUESTOR              REQUESTEDDURATION   CONDITION
my-svc.my-namespace   20m   example.com/serving   yourname@example.com   <none>              Approved,Issued
```

Download the certificate and use it

Now, as the requesting user, 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 populate `server.crt` and `server-key.pem` in a Secret that you could later mount into a Pod (for example, to use with a webserver that serves HTTPS).

```
kubectl create secret tls server --cert server.crt --key server-key.pem
```

```
secret/server created
```

Finally, you can populate `ca.pem` into a ConfigMap and use it as the trust root to verify the serving certificate:

```
kubectl create configmap example-serving-ca --from-file ca.crt=ca.pem
```

```
configmap/example-serving-ca created
```

Approving CertificateSigningRequests

A Kubernetes administrator (with appropriate permissions) can manually approve (or deny) CertificateSigningRequests 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.

Caution:

The ability to approve CSRs decides who trusts whom within your environment. The ability to approve CSRs should not be granted broadly or lightly.

You should make sure that you confidently understand both the verification requirements that fall on the approver **and** the repercussions of issuing a specific certificate before you grant the `approve` permission.

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.

For more information on certificate approval and access control, read the Certificate Signing Requests reference page.

Configuring your cluster to provide signing

This page assumes that a signer is set up 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.

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. For more information including a complete list of kubectl operations, see the `kubectl` reference documentation.

kubectl is installable on a variety of Linux platforms, macOS and Windows. Find your preferred operating system below.

- Install kubectl on Linux
- Install kubectl on macOS
- Install kubectl on Windows

kind

`kind` lets you run Kubernetes on your local computer. This tool requires that you have either Docker or Podman installed.

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 an all-in-one or a multi-node local 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