Docs Menu

# Integrate Kubernetes cert-manager with an internal ACME CA

## About this tutorial

In this example, we'll configure Kubernetes cert-manager to get a certificate from an internal ACME server, using cert-manager's ACME issuer.

- Estimated effort: Reading time ~4 mins, Lab time ~20 to 60 mins.

> If you run into any issues please let us know in GitHub Discussions.

## Requirements

- **Open source -** You have initialized and started up a `step-ca` ACME instance using the steps in our ACME server tutorial.
- **Smallstep Certificate Manager -** this tutorial assumes you have created a hosted or linked authority and created an ACME provisioner with External Account Binding enabled.
- You'll need the root certificate PEM file for your CA.

### 0. BEFORE YOU BEGIN

This example uses the ACME `dns-01` challenge type, with Google Cloud DNS. We'll create a service account on Google Cloud that cert-manager will use to solve DNS challenges. For other DNS providers, or other ACME challenge types, you'll need to change the challenge solver settings below.

### 1. CREATE A KUBERNETES CLUSTER

For this tutorial, I created a Google Compute Engine VM running a kind cluster. I'm using kind for testing, but pretty much any Kubernetes cluster will do.

```
$ kind create cluster
```

## 2. INSTALL CERT-MANAGER

Let's install Kubernetes cert-manager

First, install cert-manager:

```
$ kubectl apply --validate=false -f https://github.com/jetstack/cert-ma
```

## 3. CONFIGURE A CHALLENGE SOLVER

> **Not using Google Cloud Platform?** You can skip this step and configure the cert-manager `Issuer` in step 4 to use a different challenge solver. See cert-manager's documentation for `http-01` and `dns-01` solvers.

We're going to have cert-manager solve `dns-01` ACME challenges against a public Google Cloud Platform DNS zone. For this, we're going to create a Google Cloud Platform service account and import its credentials. The service account will need permission to manage DNS entries.

Let's create a Google Cloud Platform service account with the `roles/dns.admin` role. Replace the `PROJECT_ID` here with your own:

```
$ export PROJECT_ID=step-lan
$ gcloud iam service-accounts create dns01-solver \
   --project $PROJECT_ID --display-name "dns01-solver"
$ gcloud projects add-iam-policy-binding $PROJECT_ID \
   --member serviceAccount:dns01-solver@$PROJECT_ID.iam.gserviceaccount.com
   --role roles/dns.admin
```

Now import the service account's credentials as a Kubernetes secret:

```
$ gcloud iam service-accounts keys create key.json \
   --iam-account dns01-solver@$PROJECT_ID.iam.gserviceaccount.com
```

```
$ kubectl create secret generic clouddns-dns01-solver-svc-acct \
   --from-file=key.json
```

## 4. CREATE THE CERT-MANAGER ISSUER

Finally, let's create an cert-manager Issuer to perform `dns-01` ACME challenges. For this, we'll need a base64-encoded PEM file containing ACME server's CA certificate:

```
ROOT_CA=$(step ca root | base64)
```

Make a new file called `acme-issuer.yaml` :

```
apiVersion: cert-manager.io/v1
kind: Issuer
metadata:
  name: acme-issuer
spec:
  acme:
    email: carl@smallstep.com
    server: https://example.ca.smallstep.com/acme/acme/directory
    caBundle: LS0tLS1DRUdJTiBDRVJUSUZJEXAMPLE2UE11OWN4ckRNYWpQTlRTbkxCcEkxd1
    privateKeySecretRef:
      name: acme-issuer-account-key
    solvers:
    - dns01:
        cloudDNS:
          # Your Google Cloud Platform project ID:
          project: step-gcp-test
          # Your Google CloudDNS zone name we will use for DNS01 challenges:
          hostedZoneName: step-public-zone
          serviceAccountSecretRef:
            name: clouddns-dns01-solver-svc-acct
            key: key.json
```

Replace the values for `email` , `server` URL, `caBundle` , `project` and `hostedZoneName` with your own. Your Smallstep ACME endpoint typically takes the form of `https://[your CA hostname]/acme/acme/directory` .

### Optional: Enabling ACME External Account Binding (EAB)

Smallstep Certificate Manager uses ACME External Account Binding (EAB). When you get an EAB key from Smallstep, you'll need to convert it to `base64URL` before creating a

Kubernetes secret for it:

```
echo 'yEZNEXAMPLEnu43wV/LNZYjL3ezwnd+GOd01TaID0EE=' | sed -e 's/+/-/g'
```

Output:

```
yEZNEXAMPLEnu43wV_LNZYjL3ezwnd-GOd01TaID0EE
```

Add this secret to Kubernetes:

```
kubectl create secret generic eab-secret --from-literal \
  secret=yEZNEXAMPLEnu43wV_LNZYjL3ezwnd-GOd01TaID0EE
```

Next, see cert-manager's documentation for details on configure your EAB key and secret in your `Issuer`.

## 5. APPLY YOUR ISSUER

Finally, apply your `Issuer` configuration:

```
$ kubectl apply -f acme-issuer.yaml
```

You now have an automated ACME certificate manager running inside your Kubernetes cluster.

## 6. ISSUE A TEST CERTIFICATE

Let's get a test certificate from our ACME CA, using a Certificate object. Create a file called `tls-certificate.yaml`:

```
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: k8s-internal
  namespace: default
spec:
  secretName: k8s-internal-tls
  issuerRef:
    name: acme-issuer
```

```
  dnsNames:
  - k8s.smallstep.internal
```

Replace the `dnsNames` value with a DNS name that's inside your zone.

Apply it:

```
$ kubectl apply -f tls-certificate.yaml
```

You can check the status with `kubectl get certificaterequest` or `kubectl describe certificate`:

```
$ kubectl get certificaterequest
NAME                    READY   AGE
k8s-internal-nzbnm      True    7s
$ kubectl describe certificate k8s-internal
Name:         k8s-internal
Namespace:    default
...
Kind:         Certificate
Metadata:
  Creation Timestamp:  2020-11-03T23:06:46Z
...
Spec:
  Dns Names:
    k8s.smallstep.internal
  Issuer Ref:
    Name:       acme-issuer
  Secret Name:  k8s-internal-tls
Status:
  Conditions:
    Last Transition Time:  2020-11-03T23:11:01Z
    Message:               Certificate is up to date and has not expired
    Reason:                Ready
    Status:                True
    Type:                  Ready
  Not After:               2020-11-04T23:11:01Z
  Not Before:              2020-11-03T23:11:01Z
  Renewal Time:            2020-11-04T15:11:01Z
  Revision:                1
Events:
  Type    Reason     Age    From          Message
  ----    ------     ----   ----          -------
  Normal  Issuing    10m    cert-manager  Issuing certificate as Secret does
  Normal  Generated  10m    cert-manager  Stored new private key in temporar
```

```
  Normal  Requested  10m    cert-manager  Created new CertificateRequest res
  Normal  Issuing    9m33s  cert-manager  The certificate has been successfu
```

As you can see, cert-manager will automatically renew the certificate when approximately 2/3 of its lifetime has elapsed.

That's it! You now have automated, short-lived certificates for your Kubernetes cluster. There are many use cases for X.509 certificates issued through cert-manager.

step-ca

Tutorials

Step command reference

**Open Source**

step-ca

Step CLI

**About**

About

Support

Status

Careers

**Privacy**

Terms of use

Privacy Policy

Privacy Center

Security