EVERTRUST

Horizon Issuer

Version 1, 2026-02-11

Table of Contents

Chapter 1. Overview

horizon-issuer is an external issuer for cert-manager that allows to issue certificates through Horizon, EVERTRUST's CLM. It relies on cert-manager, the industry standard for managing certificates in cloud-native environments to operate.

This usage of cert-manager brings multiple advantages:

- Seamless integration into most workflows, since cert-manager is already the standard for requesting certificates;
- Includes battle-tested features such as automatic certificate renewal, secret management, and more;
- Benefit from cert-manager's large ecosystem of integrations with other tools and platforms.

It interacts as a gateway between cert-manager and Horizon, handling `CertificateRequests` from cert-manager without having to access secrets such as private keys, which never leave the cluster:

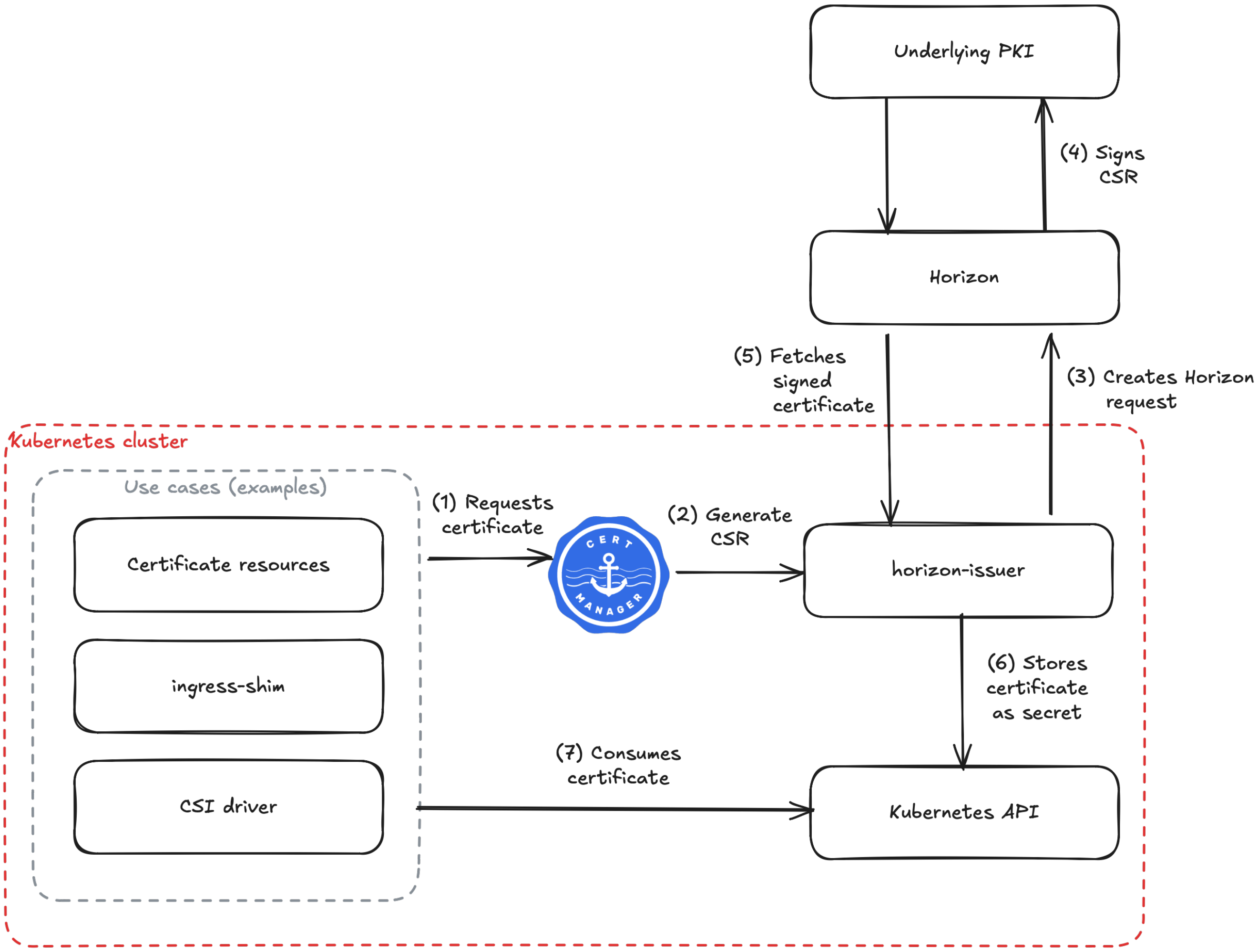

Figure 1. Overview diagram

Chapter 2. Installation

Prerequisites

Before installing, ensure the following prerequisites are met :

- Have a Kubernetes or OpenShift cluster in a version that’s supported by cert-manager;
- Have helm on a device with administrative access to the cluster (required to install CRDs);
- The cluster can either pull images from Internet or a proxy has been set up for registry.evertrust.io;
- Have access to an Horizon instance in a supported Horizon version.

Install

Install cert-manager

As horizon-issuer is a cert-manager issuer, cert-manager must be installed in the cluster before installing horizon-issuer. If not already installed, head to the Releases section on the cert-manager website and find the latest version.

Helm

```
$ helm install \
  cert-manager oci://quay.io/jetstack/charts/cert-manager \
  --version <version> \ ①
  --namespace cert-manager \
  --create-namespace \
  --set crds.enabled=true
```

① Replace with the version to install.

For details about configurable settings, head to Artifact Hub.

YAML manifests

```
$ kubectl apply -f https://github.com/cert-manager/cert-
manager/releases/download/<version>/cert-manager.yaml
```

Replace `<version>` with the version to install.

Other installation methods (such as `cmctl`) are available on the cert-manager installation documentation.

Install horizon-issuer

Similarly to cert-manager, we offer either a Helm chart or YAML manifests to install horizon-issuer:

Helm

Add the EVERTRUST Helm repository:

```
$ helm repo add evertrust https://repo.evertrust.io/repository/charts
```

Then, install the chart:

```
$ helm install horizon-issuer evertrust/horizon-issuer -- version <version>
```

> *CRD considerations*
>
> `horizon-issuer` needs CRDs to properly work. Similarly to what cert-manager offers, you have two options when installing the chart :
>
> - Manage CRDs manually: manually install the CRDs using `kubectl`. In that case, the following commands before installing or upgrading the chart:
>
>   ```
>   kubectl apply -f
>   https://raw.githubusercontent.com/evertrust/horizon-
>   issuer/v0.3.0/charts/horizon-
>   issuer/crds/horizon.evertrust.io_clusterissuers.yaml
>   kubectl apply -f
>   https://raw.githubusercontent.com/evertrust/horizon-
>   issuer/v0.3.0/charts/horizon-
>   issuer/crds/horizon.evertrust.io_issuers.yaml
>   ```
>
>   This ensures that the CRDs are not upgraded by mistake. However, it requires you to manually upgrade the CRDs when a new version is released. If you opt for this method, ensure that the `installCRDs` key is set to `false` in your Helm.
>
> - Let the Helm chart manage CRDs: in that case, the CRDs will be installed and upgraded automatically when installing or upgrading the chart. To do so, ensure that the `installCRDs` key is set to `true` in your Helm.

YAML manifests

```
$ kubectl apply -f https://github.com/evertrust/horizon-
issuer/releases/download/<version>/install.yaml
```

Upgrade

Unless noted otherwise in the below upgrade notes, upgrading horizon-issuer is not expected to break existing behaviors. horizon-issuer having reached 1.0.0, we'll follow semantic versioning and denote any breaking changes by releasing a new major version.

Helm

Update the EVERTRUST Helm repository:

```
$ helm repo update evertrust
```

Then, install the chart:

```
$ helm upgrade horizon-issuer evertrust/horizon-issuer --version <version>
```

YAML manifests

Simply apply the new manifests which should update components:

```
$ kubectl apply -f https://github.com/evertrust/horizon-
issuer/releases/download/<version>/install.yaml
```

Specific upgrade notes

Upgrade from v0.3 to v1.0

The Helm Chart has been completely reworked to be generated from the YAML manifests. Therefore, many keys in the `values.yaml` files have changed. See the reference values.yaml files to check with your local configuration for updates. Namely:

- `installCRDs` has been renamed to `crd.enable` and set to `true` by default;
- the `image` block has been moved to `manager.image`

Additionally, the already deprecated CRD version `v1alpha1` has been removed in favor of the current `v1beta1`.

Upgrade from v0.2.0 to v0.3.0

In 0.3.0, the CRDs can be managed by the Helm chart itself, similarly to what cert-manager offers. It means that you have two options when upgrading.

Should you decide to manage CRDs automatically through the Helm chart, you'll need to update existing CRDs before upgrading so that they can be managed by the Helm chart. The following commands are required :

```
$ kubectl label crd/clusterissuers.horizon.evertrust.io app.kubernetes.io/managed-by=Helm
$ kubectl label crd/issuers.horizon.evertrust.io app.kubernetes.io/managed-by=Helm
```

```
$ kubectl annotate crd/clusterissuers.horizon.evertrust.io meta.helm.sh/release-name=<horizon-issuer> meta.helm.sh/release-namespace=<horizon-issuer>
$ kubectl annotate crd/issuers.horizon.evertrust.io meta.helm.sh/release-name=<horizon-issuer> meta.helm.sh/release-namespace=<horizon-issuer>
```

Replace replace `release-name` with your Helm release name and `release-namespace` with the namespace you’re installing into.

Upgrade from v0.1.0 to v0.2.0

In 0.2.0, the new CRD version is `v1beta1`, and `v1alpha1` is no longer supported. To migrate from the old version, you must first upgrade the CRDs:

```
$ kubectl apply -f https://raw.githubusercontent.com/evertrust/horizon-issuer/v0.2.0/charts/horizon-issuer/crds/horizon.evertrust.io_clusterissuers.yaml
$ kubectl apply -f https://raw.githubusercontent.com/evertrust/horizon-issuer/v0.2.0/charts/horizon-issuer/crds/horizon.evertrust.io_issuers.yaml
```

This will not delete your existing `Issuer` and `ClusterIssuer` objects, but will allow you to create resources with the new `v1beta1` version. After having re-created your issuer objects, you can start the upgrade using Helm :

```
$ helm upgrade horizon-issuer evertrust/horizon-issuer
```

Uninstall

Helm

Simply uninstall the chart:

```
$ helm uninstall horizon-issuer
```

YAML manifests

Simply delete the manifests using `kubectl`:

```
$ kubectl delete -f https://github.com/evertrust/horizon-issuer/releases/download/<version>/install.yaml
```

Uninstalling horizon-issuer will delete any `Issuer` or `ClusterIssuer` resources created in the cluster. Make sure to backup the configuration properly before uninstalling.

Chapter 3. Configuration

Before issuing certificates, you’ll have to configure an Issuer or a ClusterIssuer resource. Issuers are a cert-manager concept that describe the entity that will be used to sign certificates. In horizon-issuer, an issuer references the following information:

- the URL of an Horizon instance;
- a reference to a secret with credentials for the Horizon instance;
- the name of the profile that will be used to issue certificates;
- other settings such as metadata and optional behaviors.

You can find more information about issuers in the cert-manager documentation. Two types of issuers are supported:

- an `Issuer` object, which provides a namespace-scoped issuer. This is the best choice if you provide namespaces-as-a-service for your users and want to restrict what they can do with the issuer (for instance to allow them enrolling certificates on a single profile), or want them to manage their issuers themselves;
- a `ClusterIssuer` object, which provides a cluster-scoped issuer. This is the best choice if you want to provide a single issuer for the entire cluster, allowing users to simply consume the configured issuer.

Either type of issuer support the same configuration parameters described below.

Configure Horizon

Depending on your desired authorization workflow, you’ll need to configure:

- a WebRA profile on Horizon;
- a principal (user or X509 certificate) with the ability to:
  - enroll certificates on the profile, either through the "Enroll" or "Request enroll" permission. In the latter case, certificates requests will be submitted for approval and marked as "Pending" in Kubernetes until approved or denied in Horizon.
  - renew certificates on the profile through the "Renew" permission;
  - optionally, revoke certificates on the profile through the "Revoke" permission, if you want to revoke certificates;

After granting the correct authorizations, provide your Horizon credentials in a secret you may create with :

```
$ kubectl create secret generic horizon-credentials \
  --from-literal=username=<horizon username> \
  --from-literal=password=<horizon password>
```

Alternatively, to authenticate using an X509 certificate, use a `kubernetes.io/tls` Secret instead:

```
$ kubectl create secret tls horizon-credentials \
  --cert=path/to/tls.crt \
  --key=path/to/tls.key
```

Provision an issuer

You’re ready to create an `Issuer` or `ClusterIssuer` object depending on the scope you want to issue certificates on :

clusterissuer.yaml

```
apiVersion: horizon.evertrust.io/v1beta1
kind: ClusterIssuer
metadata:
    name: horizon-clusterissuer ①
spec:
    url: <horizon instance URL> ②
    authSecretName: horizon-credentials ③
    profile: IssuerProfile ④
```

① Name of the ClusterIssuer object, used to reference it in Certificate resources.

② URL of the Horizon instance to connect to.

③ Name of the secret containing Horizon credentials.

④ Name of the profile to use when issuing certificates.

Then, apply the issuer manifest with :

```
$ kubectl apply -f clusterissuer.yaml
```

Additional configuration

You may provide additional configuration options in the issuer spec.

Trust custom CAs

Your Horizon instance may be presenting a certificate issued by your custom CA. To trust that certificate, you may specify a CA bundle when creating the issuer through the `caBundle` field. You may also completely disable TLS verification by setting `skipTLSVerify` to `true`, this is however highly discouraged.

Example :

```
apiVersion: horizon.evertrust.io/v1beta1
kind: ClusterIssuer
```

```
spec:
  caBundle: |
    -----BEGIN CERTIFICATE-----
    ...
    -----END CERTIFICATE-----
  skipTLSVerify: false
```

You can also mount your custom `/etc/ssl/certs` directory if you wish to have more control over the underlying OS trust store.

Revoke deleted certificates

By default, Horizon issuer does not revoke certificates deleted from Kubernetes as cert-manager can reuse the private key kept in the deleted certificate's secret. If you want to revoke certificates are they are deleted, set the `revokeCertificates` property to `true` on your `Issuer` or `ClusterIssuer` object :

```
apiVersion: horizon.evertrust.io/v1beta1
kind: ClusterIssuer
spec:
  revokeCertificates: true
```

When doing so, you may want to clean up secrets as soon as certificates are revoked.

Use an outbound proxy

If you need to use an outbound proxy to reach your Horizon instance, you may specify it in the `proxy` field of your `Issuer` or `ClusterIssuer` object :

```
apiVersion: horizon.evertrust.io/v1beta1
kind: ClusterIssuer
spec:
  proxy: http://proxy.example.com:8080
```

Validate the certificate FQDN

In case you want to enforce the coherence of your infrastructure, we offer a DNS validation feature. When enabled, the issuer will check that a DNS entry matching the certificate CN and every DNS SANs exist. If not, the certificate will not be issued. To enable, add the following key to your `Issuer` object :

```
apiVersion: horizon.evertrust.io/v1beta1
kind: ClusterIssuer
spec:
  dnsChecker:
    server: 8.8.8.8:53
```

Chapter 4. Usage

Now that your issuer is set up, you may reference it when issuing new certificates. This can be done by setting the `issuerRef` key on that certificate :

```yaml
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: demo-cert
spec:
  commonName: demo.org
  secretName: demo-cert
  issuerRef:
    group: horizon.evertrust.io
    kind: ClusterIssuer
    name: horizon-clusterissuer
```

Of course, issuing certificates manually is not the usual way of requesting certificates through cert-manager. You can tap into the long list of integrations provided out of the box by cert-manager.

For instance, if you are using `ingress-shim` to secure your ingress resources, reference your issuer using the following annotations when creating your ingress :

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: demo-ingress
  annotations:
    cert-manager.io/issuer-group: horizon.evertrust.io
    cert-manager.io/issuer-kind: ClusterIssuer
    cert-manager.io/issuer: horizon-clusterissuer
    cert-manager.io/common-name: demo.org
```

> ⚠ Be sure to set the `cert-manager.io/common-name` annotation as by default, ingress-shim will generate certificates without any DN. This will cause errors on Horizon’s side.

Configure certificate metadata

Horizon offers useful features to categorize and better understand your certificates through metadata. You may specify metadata at multiple levels. Values get overridden in the following order of precedence:

1. Values set in the `defaultTemplate` object on an `Issuer` or `ClusterIssuer` object
2. Values set on annotations either on the `Ingress` or `Certificate` object
3. Values set in the `overrideTemplate` of an `Issuer` or `ClusterIssuer` object

Using defaultTemplate on an issuer

Default templates allows you to set default values for your certificates. These values will be used if no other value is set by the user on the resource they are issuing. On the Issuer or ClusterIssuer object, add the following key :

```yaml
apiVersion: horizon.evertrust.io/v1beta1
kind: ClusterIssuer
spec:
  profile: IssuerProfile
  url: https://you.evertrust.io
  defaultTemplate:
    owner: owner-name
    team: team-name
    contactEmail: owner-email@company.com
    labels:
      label-name1: label-value1
  authSecretName: horizon-credentials
```

On an Ingress or Certificate object

You may use the following annotations on ingresses that will be reflected onto the enrolled certificate :

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ingress-name
  annotations:
    horizon.evertrust.io/owner: owner-name
    horizon.evertrust.io/team: team-name
    horizon.evertrust.io/contact-email: owner-email@company.com
    horizon.evertrust.io/labels.label-name1: label-value1
    horizon.evertrust.io/labels.label-name2: label-value2
```

These values, if set, will take precedence over annotations on values set in the defaultTemplate key of the issuer.

Using overrideTemplate on an issuer

You may also want to ensure certain values are set on every certificate issued by a specific issuer. This can be done using the overrideTemplate key on an Issuer or ClusterIssuer object. These values will take precedence over any other value set on the issuer or on the resource being issued:

```yaml
apiVersion: horizon.evertrust.io/v1beta1
kind: ClusterIssuer
```

```
spec:
  profile: IssuerProfile
  url: https://you.evertrust.io
  overrideTemplate:
    owner: owner-name
    team: team-name
    contactEmail: owner-email@company.com
    labels:
      label-name1: label-value1
  authSecretName: horizon-credentials
```

These values, if set, will take precedence over annotations on an `Ingress` or `Certificate` object.

Chapter 5. Troubleshooting

Due to the number of components involved in the architecture, it might be hard to pinpoint why a certificate fails to issue when requested by a workload.

Check whether the certificate request was created

First, ensure that cert-manager created a `CertificateRequest` resource for the `Certificate` you created.

To do so, run the following command:

```
$ kubectl get certificaterequest -n <namespace>
```

If the certificate request is not present, check cert-manager logs to see if any error occurred when trying to create it:

```
$ kubectl logs -n <cert-manager-namespace> <cert-manager-pod> --tail=100
```

Investigate the CertificateRequest status

If the certificate request is present but not in a `Ready` state, check its status for error messages:

```
$ kubectl describe certificaterequest <certificaterequest-name> -n <namespace>
```

Look for any error messages in the `Status` section that could indicate why the request failed.

Check other services logs

If the `CertificateRequest` shows errors related to the issuer, check the logs of the horizon-issuer controller for more details:

```
$ kubectl logs -n <horizon-issuer-namespace> <horizon-issuer-pod> --tail=100
```

If it shows Horizon-related errors, check the Horizon instance logs or contact EVERTRUST support for assistance.