Documentation: How to Use the CertsController

This document provides a step-by-step guide on using the **CertsController**, a Kubernetes controller designed to manage certificates by automatically creating self-signed certificates and storing them in Kubernetes `Secret` resources.

Prerequisites

- Kubernetes cluster (version 1.16 or higher).
- The `kubectl` command-line tool is configured to interact with the cluster.
- CertsController custom resources and CRDs installed in the cluster.

---

1. Install the CertsController Custom Resource Definitions (CRDs)

CertsController relies on custom resource definitions (CRDs) to define `Certificate` objects that it manages. Make sure you install the necessary CRDs before using the controller.

To install CRDs:

```
kubectl apply -f
config/crd/bases/certscontroller.kuberastic.com_certificates.yaml
```

This command registers the custom `Certificate` resource type with the Kubernetes API.

---

2. Deploy the CertsController to Your Cluster

Once the CRDs are installed, you need to deploy the CertsController itself.

To build and deploy the controller:

```
make docker-build docker-push IMG=localhost:5001/certscontroller:latest
make deploy IMG=localhost:5001/certscontroller
```

```
kubectl apply -f rbac.yaml
```

If you need to make any changes to the controller code itself, you can rebuild and rollout restart.

To rebuild and rollout restart:

```
make docker-build docker-push IMG=localhost:5001/certscontroller:latest
kubectl rollout restart deploy --selector=control-plane=controller-manager
--namespace certscontroller-system
```

This deployment includes the necessary controller and RBAC configurations that allow CertsController to interact with the Kubernetes API.

---

3. Create a Certificate Custom Resource

CertsController watches for new or updated `Certificate` custom resources and processes them to generate self-signed certificates.

You define the certificate you want using the `Certificate` custom resource.

Example `Certificate` YAML:

```yaml
apiVersion:
certscontroller.kuberastic.com.certscontroller.kuberastic.com/v1
kind: Certificate
metadata:
  name: example-certificate
  namespace: default
spec:
  # the DNS name for which the certificate should be issued
  domain: example.kuberastic.com
  # the time until the certificate expires
  validityInMonths: 3
  # a reference to the Secret object in which the certificate is stored
  secretRef:
    name: my-certificate-secret
    namespace: default
```

Key Fields:

- **`domain`**: The primary domain name for the certificate (e.g., `mydomain.com`).
- **`dnsNames`**: An optional list of alternative domain names for the certificate.
- **`validityInMonths`**: The validity period of the certificate.
- **`secretRef`**: The name of the Kubernetes `Secret` resource that will store the generated certificate and private key.

To apply the custom resource:

```
kubectl apply -f example-certificate.yaml
```

Once the `Certificate` resource is applied, the CertsController will automatically generate a self-signed certificate and store it in a `Secret` with the name specified in `secretRef`.

---

4. Accessing the Generated Certificate

After the CertsController processes the `Certificate` resource, a Kubernetes `Secret` will be created that contains the generated certificate and private key.

To view the created secret:

```
kubectl get secret my-cert-secret -o yaml
```

This `Secret` will contain:

- **`tls.crt`**: The certificate.
- **`tls.key`**: The private key.

Example output:

```
apiVersion: v1
kind: Secret
metadata:
  name: my-cert-secret
  namespace: default
type: kubernetes.io/tls
data:
  tls.crt: <base64-encoded-certificate>
  tls.key: <base64-encoded-private-key>
```

You can reference this secret in your application deployments, Ingress objects, or any other Kubernetes resource that requires a TLS certificate.

---

5. Updating or Renewing Certificates

If you need to update or renew the certificate (e.g., by changing the `commonName`, `dnsNames`, or `duration`), simply update the `Certificate` custom resource and apply the changes.

To update the certificate:

1. Modify the `Certificate` custom resource YAML.

Apply the changes:

```
kubectl apply -f path/to/updated-certificate.yaml
```

2. The CertsController will detect the changes and regenerate the certificate, updating the associated `Secret`.

---

6. Deleting Certificates

If you want to delete the certificate, you can simply delete the `Certificate` custom resource. Please note that the certificate deletion will not result in the deletion of the secret which you can do separately.

To delete the certificate and secret:

```
kubectl delete certificate my-certificate -n default
Kubectl delete my-cert-secret -n default
```

This will also clean up the associated `Secret`.

---

7. Troubleshooting

Check Controller Logs

If you encounter issues, you can check the logs of the CertsController to debug any problems:

```
kubectl logs -n certscontroller-system
deployment/certscontroller-controller-manager
```

Look for any error messages or warnings related to certificate generation, permission issues, or custom resource reconciliation.

Common Issues

- **Missing permissions**: Ensure the `ServiceAccount` used by the controller has the correct RBAC permissions to create, list, and update `Secret` resources.
- **Invalid Certificate Spec**: Ensure the fields in the `Certificate` resource are correct. Check for any invalid or missing fields.
- **CRD Not Installed**: If you encounter errors related to the `Certificate` kind not being recognized, ensure the CRD has been installed correctly.

---

8. Uninstalling the CertsController

To uninstall the CertsController and remove its associated resources from your cluster:

1. Delete all Certificate resources:

```
kubectl delete certificates --all -n <namespace>
```

2. **Delete the CertsController deployment and CRDs:**

```
kubectl delete -f config/manager/manager.yaml
kubectl delete -f
config/crd/bases/certscontroller.kuberastic.com_certificates.yaml
```

This will remove the CertsController and its CRDs from your cluster.

Conclusion

The CertsController simplifies the management of self-signed certificates within your Kubernetes cluster. By creating and watching `Certificate` custom resources, the controller automatically generates and manages certificates, storing them securely in Kubernetes `Secret` resources. Follow the steps in this documentation to set up, use, and troubleshoot the controller.