Exercise - Enable SSL/TLS on the front-end ingress

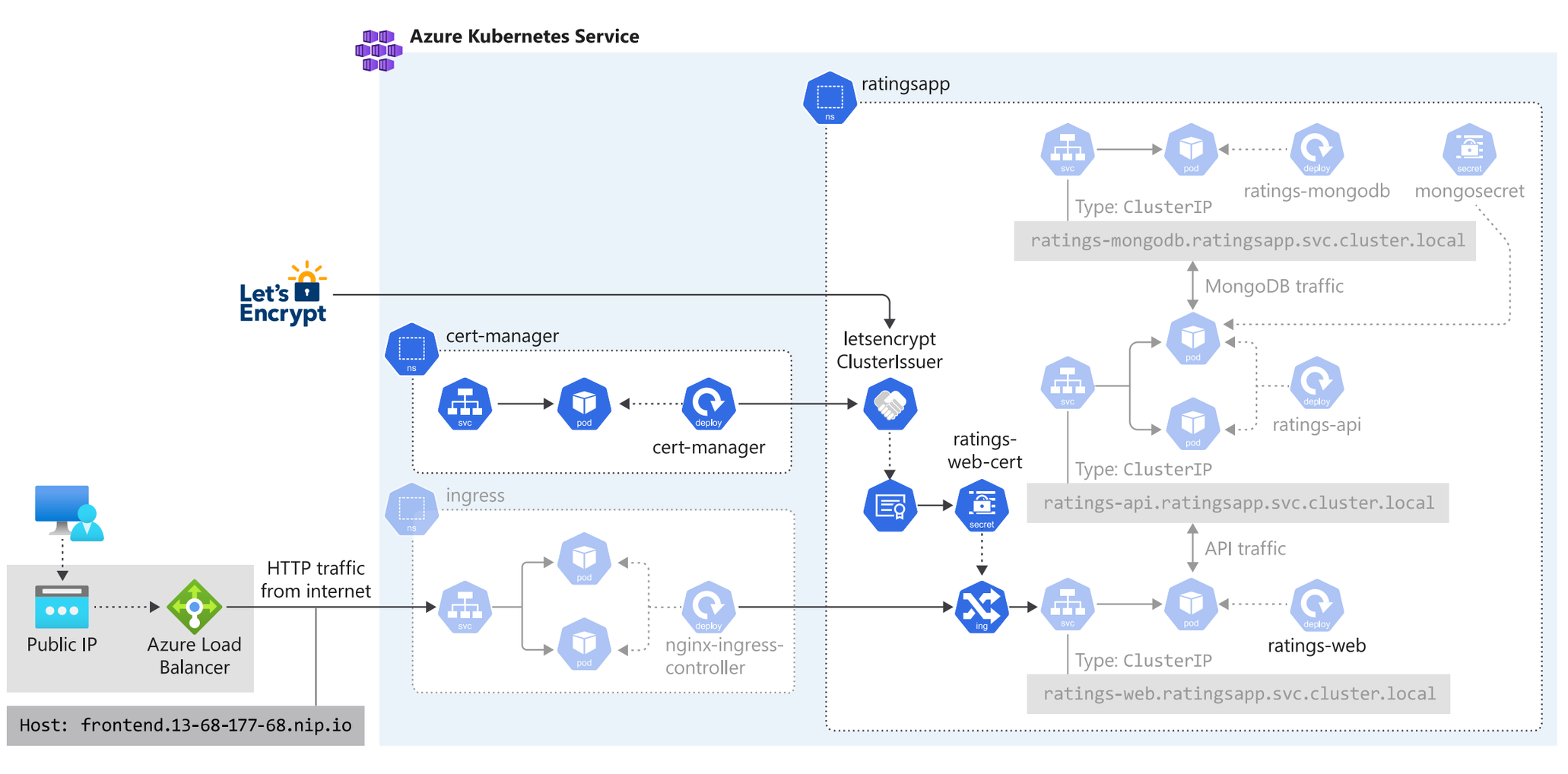
100 XP

15 minutes

The online security and privacy of user data is a primary concern for Fruit Smoothies as a company. It's important the ratings website allows HTTPS connections to all customers. NGINX ingress controller supports TLS termination and provides several ways to retrieve and configure certificates for HTTPS. This exercise demonstrates how to use *cert-manager*, which provides automatic *Let's Encrypt* certificate generation and management functionality.

In this exercise, you will:

- Deploy cert-manager by using Helm
- Deploy a ClusterIssuer resource for Let's Encrypt
- Enable SSL/TLS for the ratings web service on Ingress
- Test the application



Deploy cert-manager

cert-manager is a Kubernetes certificate management controller that makes it possible to automate certificate management in cloud-native environments. cert-manager supports various sources including Let's Encrypt, HashiCorp Vault, Venafi, simple signing key pairs, or self-signed certificates. You'll use `cert-manager` to ensure your website's certificate is valid and up to date, and attempt to renew certificates at a configured time before the certificate expires.

cert-manager uses Kubernetes custom resources. A Kubernetes custom resource is an object that allows you to extend the Kubernetes API or to introduce your API into a cluster. You use custom resource definition (CRD) files to define your object kinds and the API Server manage the lifecycle of the object.

Here, you'll use Helm to install cert-manager and then configure it to use Let's Encrypt as the certificate issuer.

1. Let's start by creating a namespace for cert-manager.

   bash Copy
   ```bash
   kubectl create namespace cert-manager
   ```

2. You'll use the Jetstack Helm repository to find and install cert-manager. First, you'll add the Jetstack Helm repository by running the code below.

   bash Copy
   ```bash
   helm repo add jetstack https://charts.jetstack.io
   helm repo update
   ```

3. Next, run the following command to install cert-manager by deploying the cert-manager CRD.

   bash Copy
   ```bash
   kubectl apply --validate=false -f https://raw.githubusercontent.com/jetstack/cert-manager/releas
   ```

4. Install the cert-manager Helm chart

   bash Copy
   ```bash
   helm install cert-manager \
       --namespace cert-manager \
       --version v0.14.0 \
       jetstack/cert-manager
   ```

5. You'll see output similar to the example below when the installation completes.

   output Copy
   ```
   NAME: cert-manager
   LAST DEPLOYED: Tue Jan  7 13:11:19 2020
   NAMESPACE: cert-manager
   STATUS: deployed
   REVISION: 1
   TEST SUITE: None
   NOTES:
   cert-manager has been deployed successfully!
   ```

6. Verify the installation by checking the `cert-manager` namespace for running pods.

   bash Copy
   ```bash
   kubectl get pods --namespace cert-manager
   ```

   You'll see that the `cert-manager`, `cert-manager-cainjector`, and `cert-manager-webhook` pod is in a `Running` state. It might take a couple of minutes to provision the web hook required for the TLS assets.

   output Copy
   ```
   NAME                                       READY   STATUS    RESTARTS   AGE
   cert-manager-5c6866597-zw7kh               1/1     Running   0          2m
   cert-manager-cainjector-577f6d9fd7-tr77l   1/1     Running   0          2m
   cert-manager-webhook-787858fcdb-nlzsq      1/1     Running   0          2m
   ```

Deploy a ClusterIssuer resource for Let's Encrypt

Cert-manager will ensure that your website's certificate is valid and up to date, and even attempt to renew certificates at a configured time before the certificate expires. However, you need to set up a *ClusterIssuer* before you can begin the certificate issuing process. The cluster issuer acts as an interface to a certificate-issuing service such as Let's Encrypt.

Let's Encrypt is a nonprofit Certificate Authority that provides TLS certificates. Let's Encrypt allows you to set up an HTTP server and have it automatically obtain a browser-trusted certificate. The process of retrieving and installing a certificate is fully automated without human intervention and managed by running a certificate management agent on the webserver. For more information about Let's Encrypt, see the *learn more* section at the end of this module.

1. Edit the file called `cluster-issuer.yaml` by using the integrated editor.

   bash Copy
   ```bash
   code cluster-issuer.yaml
   ```

2. Replace the existing content in the file with the following text. Note the change of the service `type` to `ClusterIP`.

   YAML Copy
   ```yaml
   apiVersion: cert-manager.io/v1alpha2
   kind: ClusterIssuer
   metadata:
     name: letsencrypt
   spec:
     acme:
       server: https://acme-v02.api.letsencrypt.org/directory
       email: <your email> # IMPORTANT: Replace with a valid email from your organization
       privateKeySecretRef:
         name: letsencrypt
       solvers:
       - http01:
           ingress:
             class: nginx
   ```

   In the `email` key, you'll update the value by replacing `<your email>` with a valid certificate administrator email from your organization.

3. To save the file, select `Ctrl+S`. To close the editor, select `Ctrl+Q`.

4. Apply the configuration by using the `kubectl apply` command. Deploy the cluster-issuer configuration in the `ratingsapp` namespace.

   bash Copy
   ```bash
   kubectl apply \
       --namespace ratingsapp \
       -f cluster-issuer.yaml
   ```

   You'll see an output similar to this example.

   output Copy
   ```
   clusterissuer.cert-manager.io/letsencrypt created
   ```

Enable SSL/TLS for the ratings web service on Ingress

The last part of the configuration is to configure the Kubernetes Ingress file for the ratings web service to enable SSL/TLS.

1. Edit the file called `ratings-web-ingress.yaml` by using the integrated editor.

   bash Copy
   ```bash
   code ratings-web-ingress.yaml
   ```

2. Replace the existing content in the file with the following text. Note the addition of the `cert-manager.io/issuer` annotation and the new `tls` section.

   YAML Copy
   ```yaml
   apiVersion: networking.k8s.io/v1beta1
   kind: Ingress
   metadata:
     name: ratings-web-ingress
     annotations:
       kubernetes.io/ingress.class: nginx
       cert-manager.io/cluster-issuer: letsencrypt
   spec:
     tls:
       - hosts:
         - frontend.<ingress ip>.nip.io # IMPORTANT: update <ingress ip> with the dashed public IP
         secretName: ratings-web-cert
     rules:
     - host: frontend.<ingress ip>.nip.io # IMPORTANT: update <ingress ip> with the dashed public I
       http:
         paths:
         - backend:
             serviceName: ratings-web
             servicePort: 80
           path: /
   ```

   In this file, update the `<ingress ip>` value in the `host` key with the *dashed* public IP of the ingress you retrieved earlier, for example, frontend.13-68-177-68.nip.io. This value allows you to access the ingress via a host name instead of an IP address.

3. To save the file, select `Ctrl+S`. To close the editor, select `Ctrl+Q`.

4. Apply the configuration by using the `kubectl apply` command. Deploy the updated Kubernetes ingress file in the `ratingsapp` namespace.

   bash Copy
   ```bash
   kubectl apply \
       --namespace ratingsapp \
       -f ratings-web-ingress.yaml
   ```

   You'll see an output similar to this example.

   output Copy
   ```
   ingress.networking.k8s.io/ratings-web-ingress configured
   ```

5. Verify that the certificate was issued.

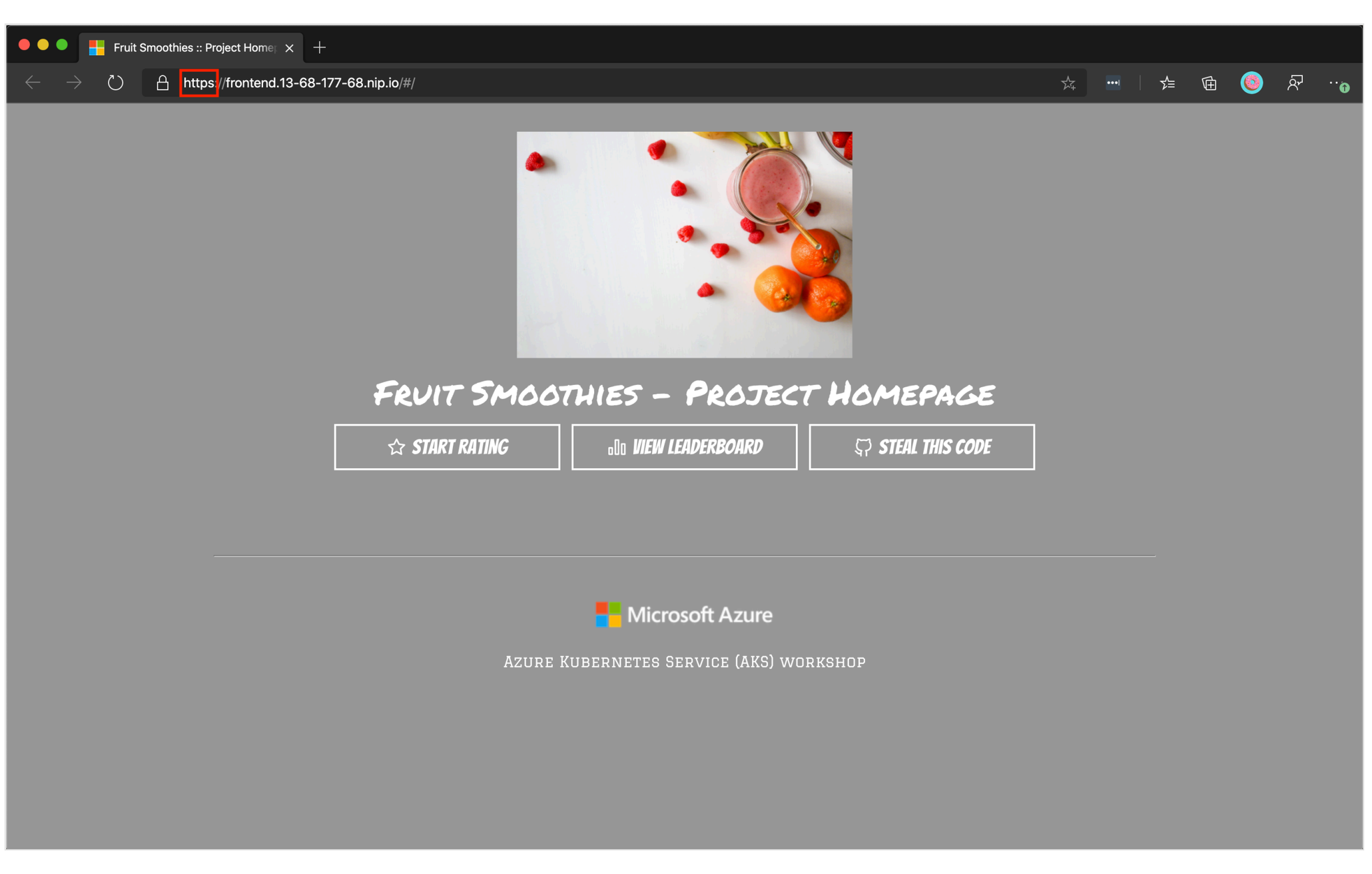
   bash Copy
   ```bash
   kubectl describe cert ratings-web-cert --namespace ratingsapp
   ```

   You'll get an output similar to this example.

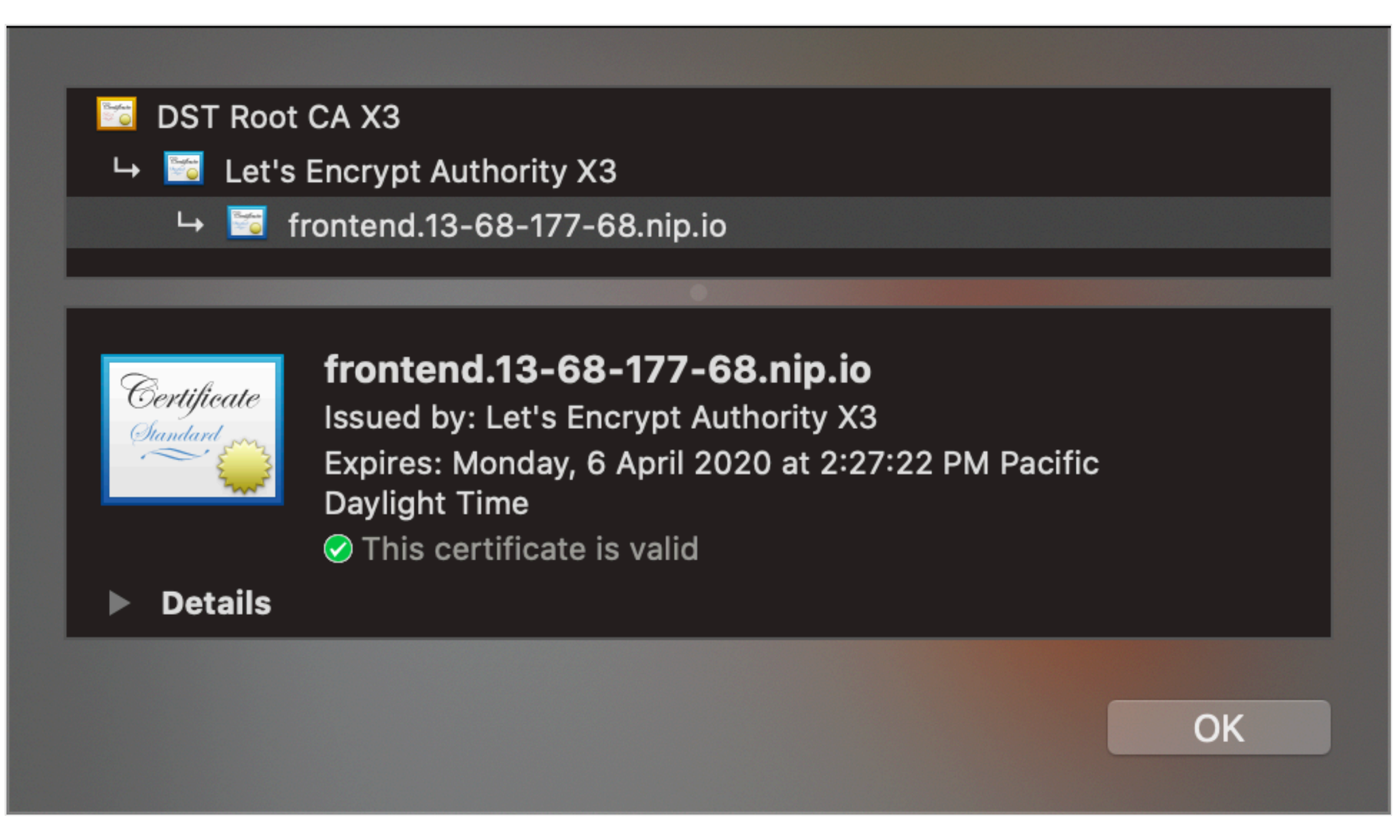
   output Copy
   ```
   Name:         ratings-web-cert
   Namespace:    ratingsapp
   API Version:  cert-manager.io/v1alpha2
   Kind:         Certificate

   [..]

   Spec:
     Dns Names:
       frontend.13-68-177-68.nip.io
     Issuer Ref:
       Group:      cert-manager.io
       Kind:       ClusterIssuer
       Name:       letsencrypt
     Secret Name:  ratings-web-cert
   Status:
     Conditions:
       Last Transition Time:  2020-01-07T22:27:23Z
       Message:               Certificate is up to date and has not expired
       Reason:                Ready
       Status:                True
       Type:                  Ready
     Not After:               2020-04-06T21:27:22Z
   Events:
     Type    Reason        Age   From          Message
     ----    ------        ----  ----          -------
     Normal  GeneratedKey  36s   cert-manager  Generated a new private key
     Normal  Requested     36s   cert-manager  Created new CertificateRequest resource "ratings-web
     Normal  Issued        34s   cert-manager  Certificate issued successfully
   ```

Test the application

Open the host name you configured on the ingress in a web browser over SSL/TLS to view and interact with the application. For example, at https://frontend.13-68-177-68.nip.io.



Verify that the front end is accessible over HTTPS and that the certificate is valid.



Summary

In this exercise, you deployed cert-manager and configured it to issue Let's Encrypt certificates automatically. You then configured the ingress you created earlier to serve encrypted TLS/SSL traffic through the generated certificates.

Next, you'll configure monitoring for your AKS cluster.

Next unit: Exercise - Configure monitoring for your application

Continue >