Lab 4.2 - Secure Ingress Traffic

Overview

In this lab, you'll be securing ingress traffic from the client for each of the three clusters by using TLS termination. Configuring TLS termination can be complicated because of the behavior of different proxies and load balancers that pass traffic to the Ingress controller, the need to negotiate TLS versions with untrusted clients, and other reasons.

Consul Cluster

1. Connect to the VM that hosts your Kubernetes clusters.

2. To start the cluster that will contain the Consul service mesh, issue this command:

```
yourname@ubuntu-vm:~$ docker start $(docker ps -a -f
name=consul-control-plane -q)

2e64e6e909e1
```

3. To switch the **kind** context to the Consul cluster, use this command:

```
yourname@ubuntu-vm:~$ kubectl config use-context kind-consul

Switched to context "kind-consul".
```


4. Forward traffic for the Emoji application locally to ensure that the app is up and running and can receive requests

```
yourname@ubuntu-vm:~$ kubectl port-forward
service/ingress-nginx-controller -n ingress-nginx 8080:80
```

5. Verify the routing requests by issuing the command below. A response code of 200 indicates that Nginx Ingress is properly routing traffic to the demo app. If you receive a different code, wait a few minutes and issue the command again.

```
yourname@ubuntu-vm:~$ curl -s -o /dev/null -w "Upstream Response
Code: %{http_code}\n" \
  http://localhost:8080

Upstream Response Code: 200
```

6. One of the most popular ways to secure Ingress Nginx traffic is by using **cert-manager**. Cert-manager is an add-on that allows you to automate the management and issuing of TLS certificates from various resources and different certificate providers. It does things like automatically ensure that the certificates you're using are up to date and renewed. In this case and for the purposes of these labs, we'll use LetsEncrypt, which is a popular and open-source way to create and utilize certificates. First we will install cert-manager, which will create everything that is needed for cert-manager to run on Kubernetes including RBAC permissions and CRDs.

```
yourname@ubuntu-vm:~$ kubectl apply -f
https://github.com/cert-manager/cert-manager/releases/download/v1.9.1/
cert-manager.yaml

namespace/cert-manager created
customresourcedefinition.apiextensions.k8s.io/certificaterequests.cert-m
anager.io created
customresourcedefinition.apiextensions.k8s.io/certificates.cert-manager.
io created
customresourcedefinition.apiextensions.k8s.io/challenges.acme.cert-manag
er.io created
customresourcedefinition.apiextensions.k8s.io/clusterissuers.cert-manage
r.io created
customresourcedefinition.apiextensions.k8s.io/issuers.cert-manager.io
created
customresourcedefinition.apiextensions.k8s.io/orders.acme.cert-manager.i
o created
```



```
serviceaccount/cert-manager-cainjector created
serviceaccount/cert-manager created
serviceaccount/cert-manager-webhook created
configmap/cert-manager-webhook created
clusterrole.rbac.authorization.k8s.io/cert-manager-cainjector created
clusterrole.rbac.authorization.k8s.io/cert-manager-controller-issuers
created
```

7. Next, you will need to create an Issuer. The Issuer is what generates the certificate and allows you to connect the cert for secure communication to your Nginx Ingress Controller. Essentially, it "issues" a certificate to your Ingress Controller so you can access your app over HTTPS. Notice how it's using an example "acme" server. The reason why is because this is a lab environment in a Kind cluster. If you were in a production environment, it would be production-level entries for server, email, etc.

```
yourname@ubuntu-vm:~$ kubectl apply -f - <<EOF
---
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
 name: letsencrypt-staging
 namespace: cert-manager
spec:
 acme:
   server: https://acme-staging-v02.api.letsencrypt.org/directory
   email: example@test.com
   privateKeySecretRef:
     name: letsencrypt-staging
   solvers:
   - http01:
       ingress:
         class:  nginx
EOF
```

8. Next, you will need to update your Ingress Controller configuration to ensure that the issuer is connected to the Ingress Controller and a Kubernetes secret is associated with the cert. The changes between the Ingress configuration below and the Ingress configuration from the previous lab are highlighted in orange so you can see the differences. You'll notice that the host is a sample "echo1.example.com". The reason why is because you're deploying this on a local KinD cluster. If this was production, it would be the path/URL to the production environment/website.

```
yourname@ubuntu-vm:~$ kubectl apply -f - <<EOF
---
```



```
    apiVersion: networking.k8s.io/v1
    kind: Ingress
    metadata:
      namespace: emojivoto
      name: ingress-emojivoto
      annotations:
        ingress.kubernetes.io/rewrite-target: /
        nginx.ingress.kubernetes.io/service-upstream: "true"
        kubernetes.io/ingress.class: istio
        consul.hashicorp.com/connect-inject: "true"
        cert-manager.io/issuer: "letsencrypt-staging"
    spec:
      tls:
      - hosts:
        - echo1.example.com
        secretName: tlstest
      ingressClassName: nginx
      rules:
        - http:
            paths:
            - pathType: Prefix
              path: "/"
              backend:
                service:
                  name: web-svc
                  port:
                    number: 80
    EOF
```

9. Now that the Ingress Controller is configured, you can forward traffic over port 443.

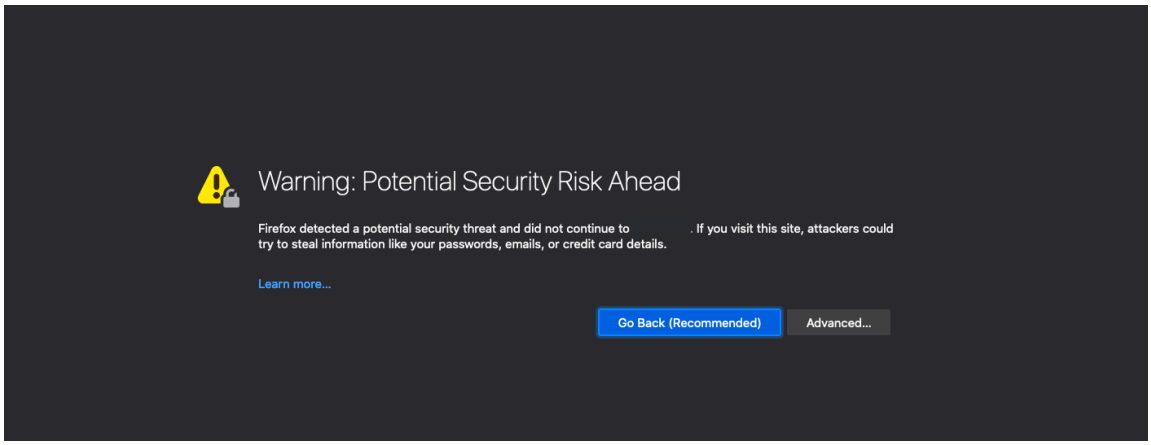
```
    yourname@ubuntu-vm:~$ kubectl port-forward
service/ingress-nginx-controller -n ingress-nginx 8080:443
```

10. Nginx Ingress is now configured to use the self-signed TLS certificate via cert-manager to establish and terminate TLS connections for any incoming host. This allows you to make requests to the VM's IP address or **`localhost`** over HTTPS. The Nginx Ingress Controller is now listening on port 443 for TLS connections instead of port 80. It is also automatically redirecting cleartext requests to HTTPS. Enter https://127.0.0.1:8080/ in a web browser and you should now see the emoji app pop up.

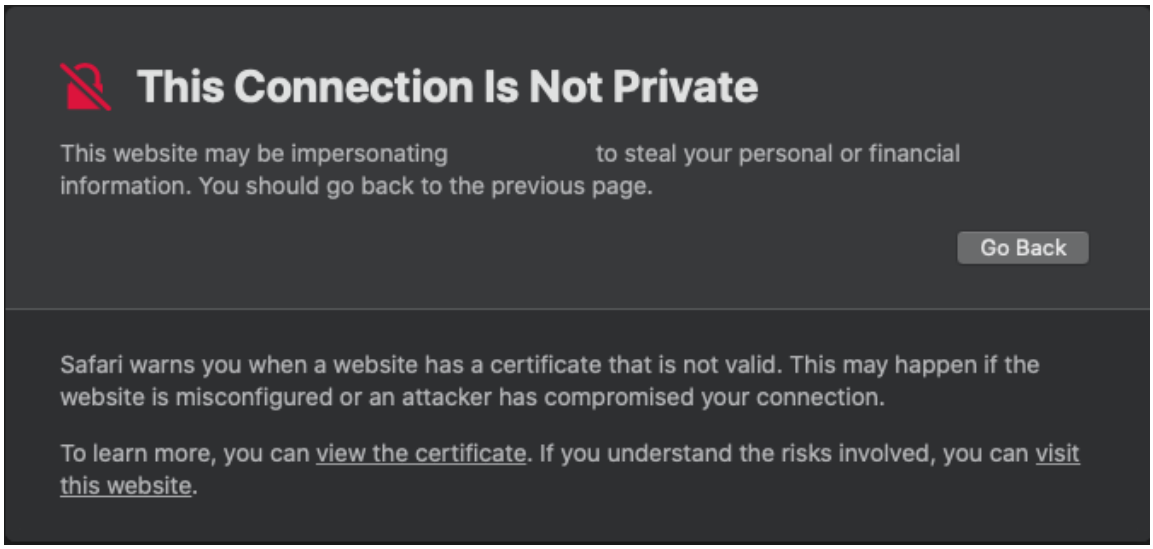
11. Since you are using a self-signed certificate, you will probably get a warning page about an untrusted certificate when trying to access the application. This is expected and can be bypassed. Below are examples of the warning you may receive from a few browsers.

If you are unable or unwilling to bypass the warning pages, you may skip this step and the next step. The warning page indicates that you are being redirected from HTTP to HTTPS and that your self-signed certificate is being encountered, and the steps you would skip are illustrating that in a second way.

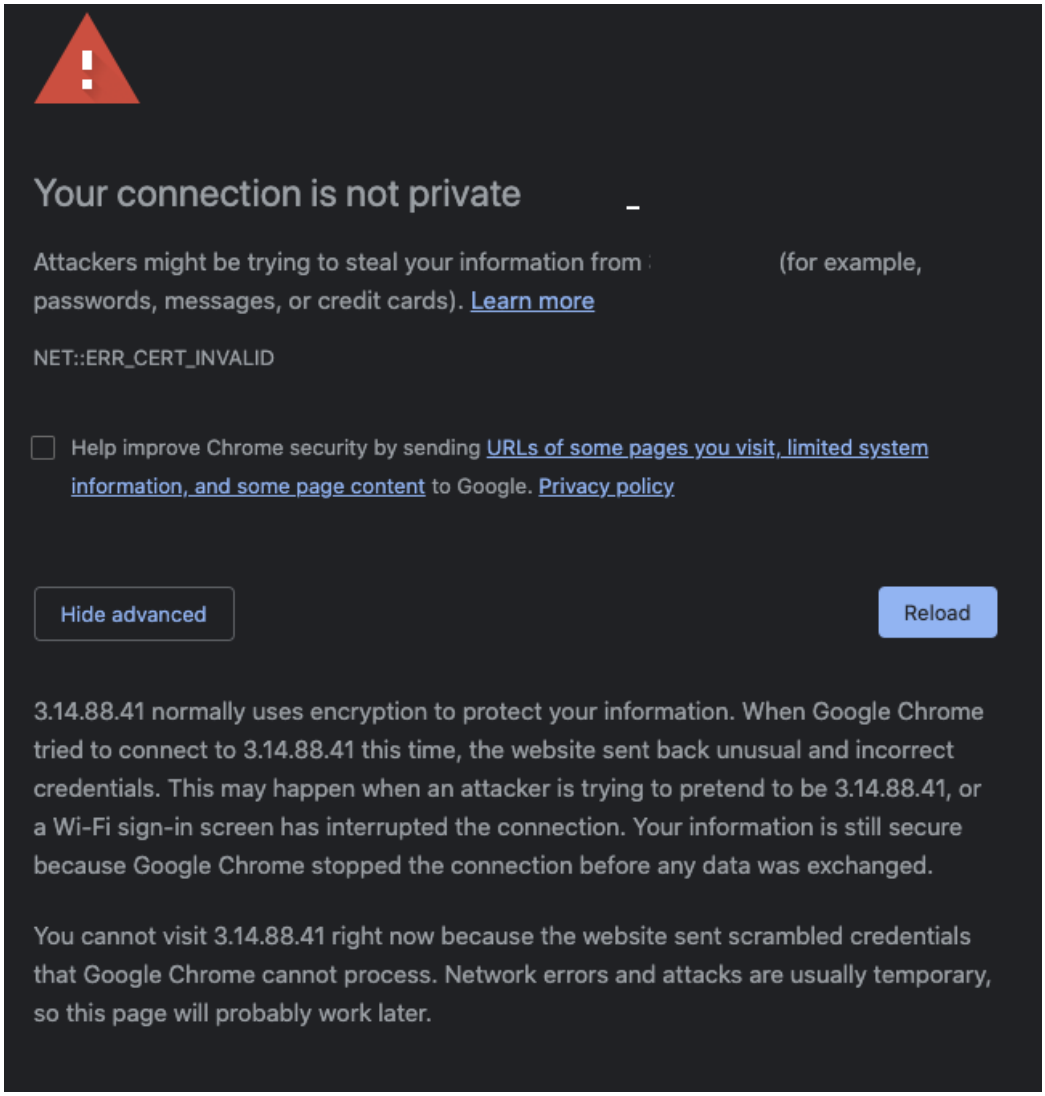
Note: If you are using Google Chrome for your browser, you will need to type **`thisisunsafe`** on the page and hit Enter or Return to bypass the security warning.



Firefox

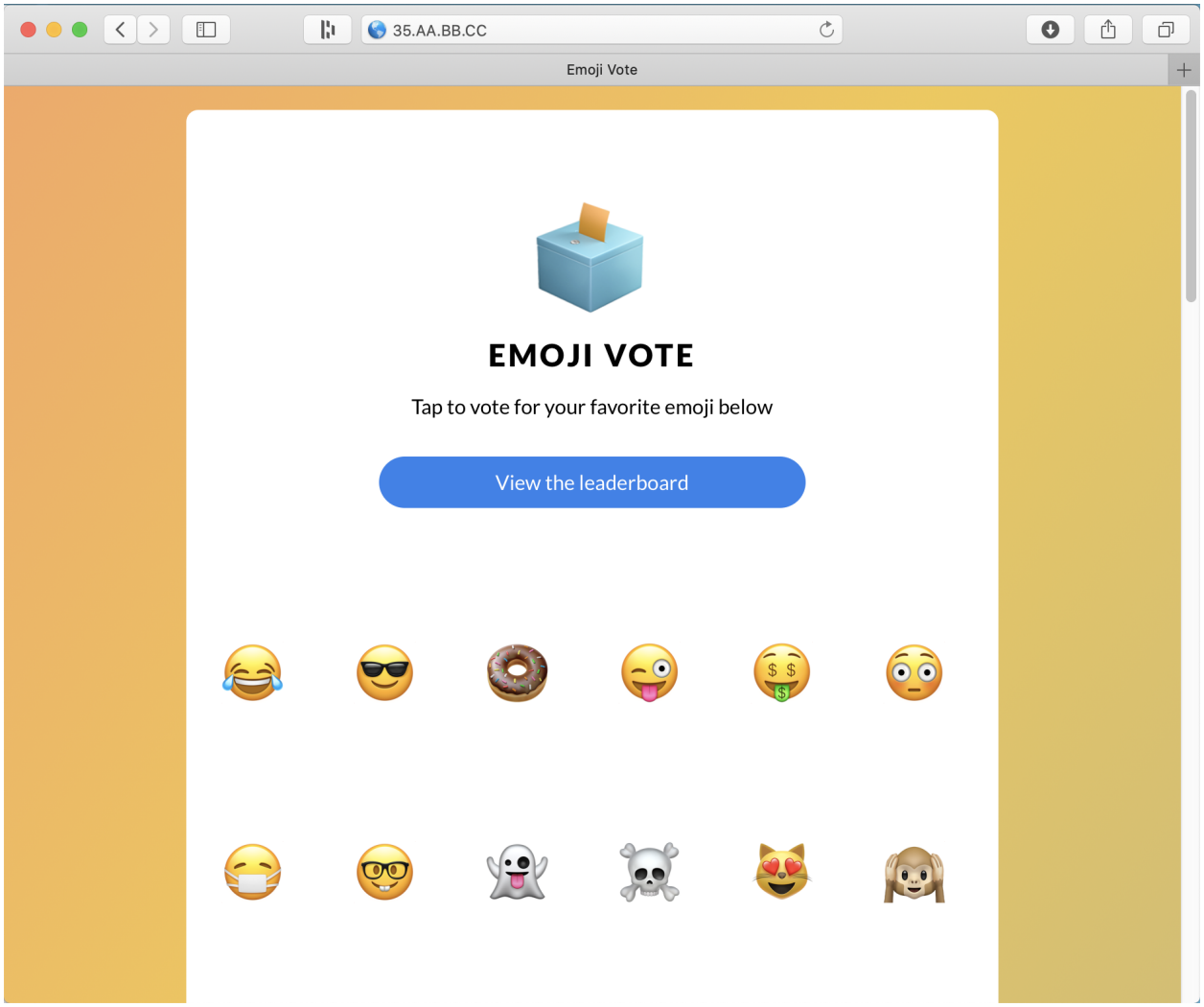


Safari



Chrome

12. After bypassing the security warning, you are now accessing the application over an encrypted connection! Try out the app. It should function just like it did in the previous lab, including giving you a 404 error when you click on the donut emoji.



13. When you are done with the app, stop the Consul cluster by using this command:

```
yourname@ubuntu-vm:~$ docker stop $(docker container ls -a -f
name=consul-control-plane -q)

2e64e6e909e1
```

Istio Cluster

14. To start the cluster that will contain the Istio service mesh, issue this command:


```
yourname@ubuntu-vm:~$ docker start $(docker ps -a -f
name=istio-control-plane -q)

2d1d09fadf21
```

15. To switch the **kind** context to the Istio cluster, use this command:

```
yourname@ubuntu-vm:~$ kubectl config use-context kind-istio

Switched to context "kind-istio".
```

16. Forward traffic for the Emoji application locally to ensure that the app is up and running and can receive requests

```
yourname@ubuntu-vm:~$ kubectl port-forward
service/emissary-ingress -n emissary 8080:80
```

17. Install Cert manager

```
yourname@ubuntu-vm:~$ kubectl apply -f
https://github.com/cert-manager/cert-manager/releases/download/v1.9.1/
cert-manager.yaml

namespace/cert-manager created
customresourcedefinition.apiextensions.k8s.io/certificaterequests.cert-m
anager.io created
customresourcedefinition.apiextensions.k8s.io/certificates.cert-manager.
io created
customresourcedefinition.apiextensions.k8s.io/challenges.acme.cert-manag
er.io created
customresourcedefinition.apiextensions.k8s.io/clusterissuers.cert-manage
r.io created
customresourcedefinition.apiextensions.k8s.io/issuers.cert-manager.io
created
customresourcedefinition.apiextensions.k8s.io/orders.acme.cert-manager.i
o created
serviceaccount/cert-manager-cainjector created
serviceaccount/cert-manager created
serviceaccount/cert-manager-webhook created
configmap/cert-manager-webhook created
clusterrole.rbac.authorization.k8s.io/cert-manager-cainjector created
```



```
clusterrole.rbac.authorization.k8s.io/cert-manager-controller-issuers
created
```

18. Create the Issuer

```
yourname@ubuntu-vm:~$ kubectl apply -f - <<EOF
---
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
 name: letsencrypt-staging
 namespace: cert-manager
spec:
 acme:
   server: https://acme-staging-v02.api.letsencrypt.org/directory
   email: example@test.com
   privateKeySecretRef:
     name: letsencrypt-staging
   solvers:
   - http01:
       ingress:
         class:  nginx
EOF
```

19. Update the ingress controller to use the cert issued by cert-manager

```
yourname@ubuntu-vm:~$ kubectl apply -f - <<EOF
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  namespace: emojivoto
  name: ingress-emojivoto
  annotations:
    ingress.kubernetes.io/rewrite-target: /
    nginx.ingress.kubernetes.io/service-upstream: "true"
    kubernetes.io/ingress.class: istio
    consul.hashicorp.com/connect-inject: “true”
    cert-manager.io/issuer: "letsencrypt-staging"
spec:
  tls:
  - hosts:
    - echo1.example.com
    secretName: tlstest
  ingressClassName: nginx
```



```
      rules:
        - http:
            paths:
            - pathType: Prefix
              path: "/"
              backend:
                service:
                  name: web-svc
                  port:
                    number: 80
    EOF
```

20. Now that the Ingress Controller is configured, you can forward traffic over port 443.

```
    yourname@ubuntu-vm:~$ kubectl port-forward
service/ingress-nginx-controller -n ingress-nginx 8080:443
```

21. Nginx Ingress is now configured to use the self-signed TLS certificate via cert-manager to establish and terminate TLS connections for any incoming host. This allows you to make requests to the VM's IP address or **`localhost`** over HTTPS. The Nginx Ingress Controller is now listening on port 443 for TLS connections instead of port 80. It is also automatically redirecting cleartext requests to HTTPS. Enter https://127.0.0.1:8080/ in a web browser and you should now see the emoji app pop up.

22. Since you are using a self-signed certificate, you will probably get a warning page about an untrusted certificate when trying to access the application. This is expected and can be bypassed.

    If you are unable or unwilling to bypass the warning pages, you may skip this step and the next step. The warning page indicates that you are being redirected from HTTP to HTTPS and that your self-signed certificate is being encountered, and the steps you would skip are illustrating that in a second way.

    Note: If you are using Google Chrome for your browser, you will need to type **`thisisunsafe`** on the page and hit Enter or Return to bypass the security warning.

23. After bypassing the security warning, you are now accessing the application over an encrypted connection! Try out the app. It should function just like it did in the previous lab, including giving you a 404 error when you click on the donut emoji.

24. When you are done with the app, stop the Istio cluster by running this command:


```
yourname@ubuntu-vm:~$ docker stop $(docker ps -a -f
name=istio-control-plane -q)

2d1d09fadf21
```

Linkerd Cluster

25. To start the cluster that will contain the Linkerd service mesh, issue this command:

```
yourname@ubuntu-vm:~$ docker start $(docker ps -a -f
name=linkerd-control-plane -q)

8af4c08524df
```

26. To switch the **kind** context to the Linkerd cluster, use this command:

```
yourname@ubuntu-vm:~$ kubectl config use-context kind-linkerd

Switched to context "kind-linkerd".
```

27. Forward traffic for the Emoji application locally to ensure that the app is up and running and can receive requests

```
yourname@ubuntu-vm:~$ kubectl port-forward
service/emissary-ingress -n emissary 8080:80
```

28. Install Cert manager

```
yourname@ubuntu-vm:~$ kubectl apply -f
https://github.com/cert-manager/cert-manager/releases/download/v1.9.1/
cert-manager.yaml

namespace/cert-manager created
customresourcedefinition.apiextensions.k8s.io/certificaterequests.cert-m
anager.io created
customresourcedefinition.apiextensions.k8s.io/certificates.cert-manager.
io created
customresourcedefinition.apiextensions.k8s.io/challenges.acme.cert-manag
er.io created
```



```
customresourcedefinition.apiextensions.k8s.io/clusterissuers.cert-manage
r.io created
customresourcedefinition.apiextensions.k8s.io/issuers.cert-manager.io
created
customresourcedefinition.apiextensions.k8s.io/orders.acme.cert-manager.i
o created
serviceaccount/cert-manager-cainjector created
serviceaccount/cert-manager created
serviceaccount/cert-manager-webhook created
configmap/cert-manager-webhook created
clusterrole.rbac.authorization.k8s.io/cert-manager-cainjector created
clusterrole.rbac.authorization.k8s.io/cert-manager-controller-issuers
created
```

29. Create the Issuer

```
yourname@ubuntu-vm:~$ kubectl apply -f - <<EOF
---
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
 name: letsencrypt-staging
 namespace: cert-manager
spec:
 acme:
   server: https://acme-staging-v02.api.letsencrypt.org/directory
   email: example@test.com
   privateKeySecretRef:
     name: letsencrypt-staging
   solvers:
   - http01:
       ingress:
         class:  nginx
EOF
```

30. Update the ingress controller to use the cert issued by cert-manager

```
yourname@ubuntu-vm:~$ kubectl apply -f - <<EOF
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  namespace: emojivoto
  name: ingress-emojivoto
  annotations:
```



```
      ingress.kubernetes.io/rewrite-target: /
      nginx.ingress.kubernetes.io/service-upstream: "true"
      kubernetes.io/ingress.class: istio
      consul.hashicorp.com/connect-inject: "true"
      cert-manager.io/issuer: "letsencrypt-staging"
  spec:
    tls:
    - hosts:
      - echo1.example.com
      secretName: tlstest
    ingressClassName: nginx
    rules:
      - http:
          paths:
          - pathType: Prefix
            path: "/"
            backend:
              service:
                name: web-svc
                port:
                  number: 80
  EOF
```

31. Now that the Ingress Controller is configured, you can forward traffic over port 443.

```
    yourname@ubuntu-vm:~$ kubectl port-forward
service/ingress-nginx-controller -n ingress-nginx 8080:443
```

32. Nginx Ingress is now configured to use the self-signed TLS certificate via cert-manager to establish and terminate TLS connections for any incoming host. This allows you to make requests to the VM's IP address or `localhost` over HTTPS. The Nginx Ingress Controller is now listening on port 443 for TLS connections instead of port 80. It is also automatically redirecting cleartext requests to HTTPS. Enter https://127.0.0.1:8080/ in a web browser and you should now see the emoji app pop up

33. Since you are using a self-signed certificate, you will probably get a warning page about an untrusted certificate when trying to access the application. This is expected and can be bypassed.

    If you are unable or unwilling to bypass the warning pages, you may skip this step and the next step. The warning page indicates that you are being redirected from HTTP to HTTPS and that your self-signed certificate is being encountered, and the steps you would skip are illustrating that in a second way.

Note: If you are using Google Chrome for your browser, you will need to type **`thisisunsafe`** on the page and hit Enter or Return to bypass the security warning.

34. After bypassing the security warning, you are now accessing the application over an encrypted connection! Try out the app. It should function just like it did in the previous lab, including giving you a 404 error when you click on the donut emoji.

35. When you are done with the app, stop the Linkerd cluster by running this command:

```
yourname@ubuntu-vm:~$ docker stop $(docker ps -a -f
name=linkerd-control-plane -q)

8af4c08524df
```

