Lab 4.1 - Deploy an Ingress Controller

Overview

In this lab, you'll be installing an ingress controller for each of the three clusters. It is the typical way of getting traffic from outside of your cluster to apps running within the cluster.

The Nginx Ingress Controller is a popular ingress controller. It has native integrations with all three service meshes we'll be using in subsequent labs—Consul, Istio, and Linkerd—so we will use it as our ingress controller throughout this course.

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. To install the Nginx Ingress Controller in the VM, use the following Kubernetes manifests which will create the namespace, install the Operator, install the CRDs, permissions, and install the Ingress.


```
yourname@ubuntu-vm:~$
kubectl apply -f
https://raw.githubusercontent.com/kubernetes/ingress-nginx/contro
ller-v1.1.1/deploy/static/provider/cloud/deploy.yaml

namespace/ingress-nginx
serviceaccount/ingress-nginx
configmap/ingress-nginx-controller
clusterrole.rbac.authorization.k8s.io/ingress-nginx
clusterrolebinding.rbac.authorization.k8s.io/ingress-nginx
role.rbac.authorization.k8s.io/ingress-nginx unchanged
rolebinding.rbac.authorization.k8s.io/ingress-nginx
service/ingress-nginx-controller-admission
service/ingress-nginx-controller
deployment.apps/ingress-nginx-controller
ingressclass.networking.k8s.io/nginx
validatingwebhookconfiguration.admissionregistration.k8s.io/ingress-ngin
x-admission
serviceaccount/ingress-nginx-admission
clusterrole.rbac.authorization.k8s.io/ingress-nginx-admission
clusterrolebinding.rbac.authorization.k8s.io/ingress-nginx-admission
role.rbac.authorization.k8s.io/ingress-nginx-admission
rolebinding.rbac.authorization.k8s.io/ingress-nginx-admission
job.batch/ingress-nginx-admission-create
job.batch/ingress-nginx-admission-patch
```

5. Make sure that all Kubernetes resources in the **`ingress-nginx`** namespace are running successfully. The resources will include Pods, Services, Deployments, and ReplicaSets.

```
yourname@ubuntu-vm:~$ kubectl get all -n ingress-nginx

pod/nginxingress-nginx-ingress-6c64f544c6-pcw89   1/1     Running
0          17s
service/nginxingress-nginx-ingress   LoadBalancer   10.96.99.199
<pending>     80:32496/TCP,443:32252/TCP   18s
pod/emissary-ingress-9c45f6447-lfbcx         1/1     Running   0
113s
deployment.apps/nginxingress-nginx-ingress   1/1     1
1           18s
replicaset.apps/nginxingress-nginx-ingress-6c64f544c6   1
1         1       18s
```


At this point, you may be wondering why the Ingress Controller was installed in the **`ingress-nginx`**. When installing Ingress Controllers, there will be several different use cases. Whether it's for a specific app, a cluster-wide ingress controller for all apps, or even an Ingress Controller that's on a separate worker node listening to all requests that come in. We've decided to go with the cluster-wide Ingress Controller option for isolation purposes. We can still have the Ingress Controller listen in on only specific namespaces, which you'll see next.

You can learn more about the different options per the Nginx Ingress documentation below:

- **Cluster-wide Ingress Controller (default)**. The Ingress Controller handles configuration resources created in any namespace of the cluster. As NGINX is a high-performance load balancer capable of serving many applications at the same time, this option is used by default in our installation manifests and Helm chart.
- **Single-namespace Ingress Controller**. You can configure the Ingress Controller to handle configuration resources only from a particular namespace, which is controlled through the **`-watch-namespace`** command-line argument. This can be useful if you want to use different NGINX Ingress Controllers for different applications, both in terms of isolation and/or operation.
- **Ingress Controller for Specific Ingress Class**. This option works in conjunction with either of the options above. You can further customize which configuration resources are handled by the Ingress Controller by configuring the class of the Ingress Controller and using that class in your configuration resources. See the section Configuring Ingress Class.

6. Next, create a Kubernetes manifest that has the Ingress API which contains the name of the Emoji app service, port, and path to reach the application. You'll notice that there are three annotations - enabling service upstream, Istio, and HashiCorp Consul. These are all for the Service Mesh's that we'll be working with throughout this course. Don't worry about them too much right now as we'll be diving into it more in Chapter 5.

```
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"
  spec:
    # ingressClassName: nginx
    rules:
      - http:
          paths:
          - pathType: Prefix
            path: "/"
            backend:
              service:
                name: web-svc
                port:
                  number: 80

EOF
```

7. Notice that the **ingressClassName** is commented out in the above code. If you keep the **ingressClass** on with the Istio annotation, you'll receive an error. However, if you apply the Ingress configuration without the **ingressClassName** and Istio turned on, and then once it's created, apply the configuration again, it'll work just fine. Run the following code to re-apply the Ingress configuration with the **ingressClassName**:

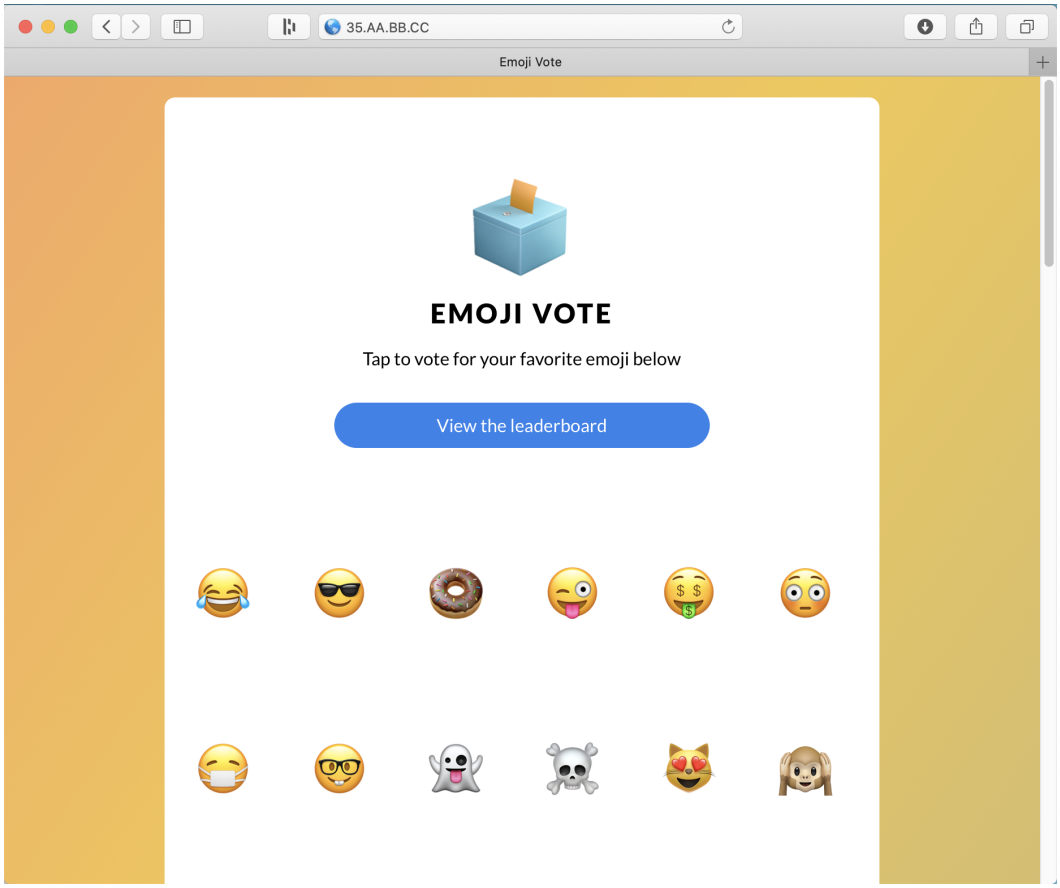
```
yourname@ubuntu-vm:~$ kubectl apply -f - <<EOF
—
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  namespace: emojivoto
  name: ingress-emojivoto
  annotations:
    ingress.kubernetes.io/rewrite-target: /
    nginx.ingress.kubernetes.io/service-upstream: "true"
    kubernetes.io/ingress.class: istio
    consul.hashicorp.com/connect-inject: "true"
spec:
  ingressClassName: nginx
  rules:
    - http:
        paths:
        - pathType: Prefix
```



```
            path: "/"
            backend:
              service:
                name: web-svc
                port:
                  number: 80
      EOF
```

8. The following command is to forward traffic from the Ingress Controller to your localhost because the Nginx Ingress Controller does not have a load balancer associated with it. The Nginx Ingress Controller listens on port 80, and you're reaching it via your localhost on port 8080.To test that the EmojiApp works, run the following command, open up a web browser, and go to **`http://127.0.0.1:8080/`**.

```
kubectl port-forward service/ingress-nginx-controller -n
ingress-nginx 8080:80
```



9. Try out the Emoji Vote app. You might notice that some parts of the app are broken—for example, if you click on the donut emoji, you'll get a 404 page. Don't worry, these errors are intentional (and we'll correct them in subsequent labs.)

10. When you are done trying out the demo 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

11. 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
```

12. 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".
```

13. To install the Nginx Ingress Controller in the VM, use the following Kubernetes Manifests which will create the namespace, install the Operator, install the CRDs, permissions, and install the Ingress:

```
yourname@ubuntu-vm:~$

kubectl apply -f
https://raw.githubusercontent.com/kubernetes/ingress-nginx/contro
ller-v1.1.1/deploy/static/provider/cloud/deploy.yaml

namespace/ingress-nginx created
serviceaccount/ingress-nginx created
configmap/ingress-nginx-controller created
clusterrole.rbac.authorization.k8s.io/ingress-nginx created
clusterrolebinding.rbac.authorization.k8s.io/ingress-nginx created
```



```
role.rbac.authorization.k8s.io/ingress-nginx created
rolebinding.rbac.authorization.k8s.io/ingress-nginx created
service/ingress-nginx-controller-admission created
service/ingress-nginx-controller created
deployment.apps/ingress-nginx-controller created
ingressclass.networking.k8s.io/nginx created
validatingwebhookconfiguration.admissionregistration.k8s.io/ingress-ngin
x-admission created
serviceaccount/ingress-nginx-admission created
clusterrole.rbac.authorization.k8s.io/ingress-nginx-admission created
clusterrolebinding.rbac.authorization.k8s.io/ingress-nginx-admission
created
role.rbac.authorization.k8s.io/ingress-nginx-admission created
rolebinding.rbac.authorization.k8s.io/ingress-nginx-admission created
job.batch/ingress-nginx-admission-create created
job.batch/ingress-nginx-admission-patch created
```

14. Make sure that all Kubernetes resources in the **ingress-nginx** namespace are running successfully. The resources will include Pods, Services, Deployments, and ReplicaSets:

```
yourname@ubuntu-vm:~$ kubectl get all -n ingress-nginx
NAME                                            READY   STATUS
RESTARTS   AGE
pod/ingress-nginx-admission-create-vcspx        0/1     Completed   0
113s
pod/ingress-nginx-admission-patch-nvggr         0/1     Completed   1
113s
pod/ingress-nginx-controller-b66cc4b74-njp2t    1/1     Running     0
113s

NAME                                         TYPE           CLUSTER-IP
EXTERNAL-IP   PORT(S)                      AGE
service/ingress-nginx-controller             LoadBalancer
10.96.182.254   <pending>     80:30635/TCP,443:31499/TCP   113s
service/ingress-nginx-controller-admission   ClusterIP      10.96.9.72
<none>        443/TCP                      113s

NAME                                       READY   UP-TO-DATE
AVAILABLE   AGE
deployment.apps/ingress-nginx-controller   1/1     1            1
113s
```



```
NAME                                                  DESIRED   CURRENT
READY   AGE
replicaset.apps/ingress-nginx-controller-b66cc4b74    1         1
1       113s

NAME                                       COMPLETIONS   DURATION
AGE
job.batch/ingress-nginx-admission-create   1/1           14s
113s
job.batch/ingress-nginx-admission-patch    1/1           15s
113s
```

15. Next, create a Kubernetes Manifest that has the Ingress API which contains the name of the Emoji app service, port, and path to reach the application. You'll notice that there are three annotations - enabling service upstream, Istio, and HashiCorp Consul. These are all for the Service Mesh's that we'll be working with throughout this course. Don't worry about them too much right now as we'll be diving into it more in Chapter 5.

```
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”
      spec:
        # ingressClassName: nginx
        rules:
          - http:
              paths:
              - pathType: Prefix
                path: "/"
                backend:
                  service:
                    name: web-svc
                    port:
```



```
                  number: 80

EOF
```

16. Notice that the ingressClassName is commented out in the above code. If you keep the ingressClass on with the Istio annotation, you'll receive an error. However, if you apply the Ingress configuration without the ingressClassName and Istio turned on, and then once it's created, apply the configuration again, it'll work just fine. Run the following code to re-apply the Ingress configuration with the ingressClassName

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

17. The following command is to forward traffic from the Ingress Controller to your localhost because the Nginx Ingress Controller does not have a load balancer associated with it. The Nginx Ingress Controller listens on port 80, and you're reaching it via your localhost on port 8080.To test that the EmojiApp works, run the following command, open up a web browser, and go to http://127.0.0.1:8080/


```
kubectl port-forward service/ingress-nginx-controller -n
ingress-nginx 8080:80
```

18. When you are done with the Emoji Vote application, 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

19. 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
```

20. 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".
```

21. To install the Nginx Ingress Controller in the VM, use the following Kubernetes Manifests which will create the namespace, install the Operator, install the CRDs, permissions, and install the Ingress.

```
yourname@ubuntu-vm:~$

kubectl apply -f
https://raw.githubusercontent.com/kubernetes/ingress-nginx/contro
ller-v1.1.1/deploy/static/provider/cloud/deploy.yaml


namespace/ingress-nginx created
serviceaccount/ingress-nginx created
configmap/ingress-nginx-controller created
clusterrole.rbac.authorization.k8s.io/ingress-nginx created
clusterrolebinding.rbac.authorization.k8s.io/ingress-nginx created
```



```
role.rbac.authorization.k8s.io/ingress-nginx created
rolebinding.rbac.authorization.k8s.io/ingress-nginx created
service/ingress-nginx-controller-admission created
service/ingress-nginx-controller created
deployment.apps/ingress-nginx-controller created
ingressclass.networking.k8s.io/nginx created
validatingwebhookconfiguration.admissionregistration.k8s.io/ingress-nginx-admi
ssion created
serviceaccount/ingress-nginx-admission created
clusterrole.rbac.authorization.k8s.io/ingress-nginx-admission created
clusterrolebinding.rbac.authorization.k8s.io/ingress-nginx-admission created
role.rbac.authorization.k8s.io/ingress-nginx-admission created
rolebinding.rbac.authorization.k8s.io/ingress-nginx-admission created
job.batch/ingress-nginx-admission-create created
job.batch/ingress-nginx-admission-patch created
```

22. Make sure that all Kubernetes resources in the ingress-nginx namespace are running successfully. The resources will include Pods, Services, Deployments, and ReplicaSets

```
kubectl get all -n ingress-nginx
```

```
NAME                                            READY   STATUS
RESTARTS   AGE
pod/ingress-nginx-admission-create-r7gvr        0/1     Completed   0
63s
pod/ingress-nginx-admission-patch-6prps         0/1     Completed   0
63s
pod/ingress-nginx-controller-b66cc4b74-pp4kh    1/1     Running     0
64s

NAME                                         TYPE           CLUSTER-IP
EXTERNAL-IP   PORT(S)                      AGE
service/ingress-nginx-controller             LoadBalancer
10.96.142.246   <pending>     80:32495/TCP,443:30997/TCP   64s
service/ingress-nginx-controller-admission   ClusterIP
10.96.228.215   <none>        443/TCP                      64s

NAME                                       READY   UP-TO-DATE
AVAILABLE   AGE
deployment.apps/ingress-nginx-controller   1/1     1            1
64s

NAME                                                  DESIRED   CURRENT
READY   AGE
replicaset.apps/ingress-nginx-controller-b66cc4b74    1         1
1       64s
```



```
NAME                                      COMPLETIONS   DURATION   AGE
job.batch/ingress-nginx-admission-create  1/1           7s         63s
job.batch/ingress-nginx-admission-patch   1/1           7s         63s
```

23. Next, create a Kubernetes Manifest that has the Ingress API which contains the name of the Emoji app service, port, and path to reach the application. You'll notice that there are three annotations - enabling service upstream, Istio, and HashiCorp Consul. These are all for the Service Mesh's that we'll be working with throughout this course. Don't worry about them too much right now as we'll be diving into it more in Chapter 5.

```
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”
      spec:
        # ingressClassName: nginx
        rules:
          - http:
              paths:
              - pathType: Prefix
                path: "/"
                backend:
                  service:
                    name: web-svc
                    port:
                      number: 80

      EOF
```

24. Notice that the ingressClassName is commented out in the above code. If you keep the **`ingressClass`** on with the Istio annotation, you'll receive an error. However, if you apply the Ingress configuration without the **`ingressClassName`** and Istio turned on, and then once it's created, apply the configuration again, it'll work just fine. Run the following code to re-apply the Ingress configuration with the **`ingressClassName`**.


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

25. The following command is to forward traffic from the Ingress Controller to your localhost because the Nginx Ingress Controller does not have a load balancer associated with it. The Nginx Ingress Controller listens on port 80, and you’re reaching it via your localhost on port 8080.To test that the EmojiApp works, run the following command, open up a web browser, and go to http://127.0.0.1:8080/

```
kubectl port-forward service/ingress-nginx-controller -n
ingress-nginx 8080:80
```

26. To stop the Linkerd cluster, run this command:

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

8af4c08524df
```

