6: Kubernetes Networking¶

Prepare Your Lab¶

Let's get your lab environment setup by automatically performing necessary steps from previous labs in this course. We've automated it for you with the following command:

```
eval "$BOOTSTRAP_COMMAND"
```

6.1: Overview¶

In a prior workshop, we exposed the gowebapp frontend using a NodePort service. In production, users will expect to access a web service on the standard http(s) ports, 80 and 443. With Kubernetes we can accomplish this with either `serviceType: LoadBalancer` (layer 4) or an `Ingress` (layer 7). In this lab, we will modify the application to expose the frontend using an Ingress Controller and Ingress Resource, which will allow users to connect to the application on 80 and 443.

6.2: Modify gowebapp Service¶

Since we will be exposing the frontend using an Ingress controller, we no longer need to expose it using a NodePort service. We still need a ClusterIP service to provide load balancing for the pods. Let's modify the service to change it to `type: ClusterIP`.

Step 01: Check the Current Service Type¶

View the `gowebapp` service and note that the service type is `NodePort`

```
kubectl get service -l "app=gowebapp"
```

Step 02: Modify gowebapp-service.yaml¶

```
cd $HOME/gowebapp/gowebapp
```

Open `gowebapp-service.yaml` in your editor and change the service type to `ClusterIP` or run the following command to make the change automatically:

```
sed -i -e 's/NodePort/ClusterIP/' gowebapp-service.yaml
```

Step 03: Update the Service¶

```
kubectl apply -f gowebapp-service.yaml
```

Step 04: Verify the service type has changed¶

```
kubectl get service -l "app=gowebapp"
```

The service type should now be `ClusterIP`.

The Ingress Controller¶

Your workshop environment's kubernetes cluster already has the Contour ingress controller installed. In addition, a specific domain name is already associated with the ingress controller's load balancer endpoint (check the value of the INGRESS_DOMAIN environment variable).

Create an Ingress Resource¶

Step 01: Define the ingress Resource¶

```
cd $HOME/gowebapp/gowebapp
```

Study the following ingress resource definition:

```
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: gowebapp-ingress
spec:
  rules:
    - host:
${SESSION_NAMESPACE}-gowebapp.${INGRESS_DOMAIN}
      http:
        paths:
          - pathType: Prefix
            path: "/"
            backend:
              service:
                name: gowebapp
                port:
                  number: 8080
```

Review the ingress specification and make sure you understand it. If you need help, please consult the reference documentation at https://kubernetes.io/docs/concepts/services-networking/ingress/

Create a file named `gowebapp-ingress.yaml` with the above contents by running the following command:

```
curl -s -o gowebapp-ingress.yaml
http://$WORKSHOP_NAMESPACE-files/_static/lab-files/gowebapp-host-ingress.yaml
```

Step 02: Apply the ingress resource¶

Notice how the ingress resource you just authored uses environment variables. The first, SESSION_NAMESPACE is used to ensure that the route to your application is unique. The second is the ingress domain that's associated with your cluster's ingress controller load balancer.

`envsubst` is a useful command that we can leverage to dynamically replace these environment variables with their values before we send the final spec to kubernetes.

Study the below command, and then run it:

```
envsubst < gowebapp-ingress.yaml | kubectl apply -f -
```

The above hyphen (`-`) tells kubectl to apply the content from standard input instead of using a file.

Step 03: Test access to gowebapp¶

Run the following command to verify the Ingress resource has been created:

```
kubectl get ingress
```

Test access to the application by accessing the host ingress URL into your browser. The URL can be shown by running the following command:

```
echo http://${SESSION_NAMESPACE}-gowebapp.${INGRESS_DOMAIN}
```