Exercise - Deploy an ingress for the front end

100 XP

20 minutes

In the previous units, you exposed the Fruit Smoothies' ratings website and RESTful API in two different ways for allowing access to each instance. The API is exposed via a ratings-api service using a *ClusterIP* that creates an internal IP address for use within the cluster. Recall, choosing this value makes the service reachable only from within the cluster. The website is exposed via a ratings-web service using a *LoadBalancer* that creates a public IP address in Azure and assigns it to Azure Load Balancer. Recall, choosing this value makes the service reachable from outside the cluster.

Even though the load balancer exposes the ratings website via a publicly accessible IP, there are limitations that you need to consider.

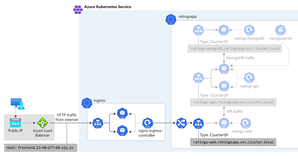
Let's assume the Fruit Smoothies' development team decides to extend the project by adding a video upload website. Fans of Fruit Smoothies can submit videos of how they're enjoying their smoothies at home, at the beach, or work. The current ratings website responds at `fruitsmoothies.com`. When you deploy the new video site, you want the new site to respond at `fruitsmoothies.com/videos` and the ratings site at `fruitsmoothies.com/ratings`.

If you continue to use the load balancer solution, you'll need to deploy a separate load balancer on the cluster and map its IP address to a new fully-qualified domain name (FQDN), for example, `videos.fruitsmoothies.com`. To implement the required URL-based routing configuration, you'll need to install additional software outside of your cluster.

The extra effort is that a Kubernetes load balancer service is a Layer 4 load balancer. Layer 4 load balancers only deal with routing decisions between IPs addresses, TCP, and UDP ports. Kubernetes provides you with an option to simplify the above configuration by using an ingress controller.

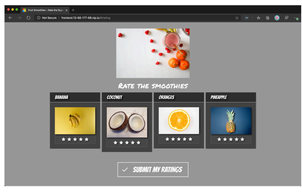
In this exercise, you will:

- Deploy a Kubernetes ingress controller running NGINX
- Reconfigure the ratings web service to use ClusterIP
- Create an ingress resource for the ratings web service
- Test the application



Deploy a Kubernetes ingress controller running NGINX

A Kubernetes ingress controller is software that provides layer 7 load balancer features. These features include reverse proxy, configurable traffic routing, and TLS termination for Kubernetes services. You install the ingress controller and configure it to replace the load balancer. With the ingress controller, you can now do all load balancing, authentication, TSL/SSL, and URL-based routing configuration without the need for extra software outside of the cluster.



There are several options for running Kubernetes ingress on Azure Kubernetes Service (AKS), such as Azure Application Gateway, Ambassador, HAProxy, Kong, NGINX, and Traefik. The ingress controllers are exposed to the internet by using a Kubernetes service of type `LoadBalancer`. The ingress controller watches and implements Kubernetes ingress resources, which create routes to application endpoints. Here, you'll deploy a basic Kubernetes ingress controller by using NGINX. Then you'll configure the ratings front-end service to use that ingress for traffic.

NGINX ingress controller is deployed as any other deployment in Kubernetes. You can either use a deployment manifest file and specify the NGINX ingress controller image or you can use an *nginx-ingress* Helm chart. The NGINX Helm chart simplifies the deployment configuration required for the ingress controller. For example, you don't need to define a configuration mapping or configure a service account for the NGINX deployment. Here, you'll use a Helm chart to install the ingress controller on your cluster.

1. Start by creating a namespace for the ingress.

   ```bash
   kubectl create namespace ingress
   ```

2. Next, install the NGINX ingress controller. NGINX ingress is part of the stable Helm repository you configured earlier when you installed MongoDB. You'll install two replicas of the NGINX ingress controllers are deployed with the `--set controller.replicaCount` parameter for added redundancy. Make sure to schedule the controller only on Linux nodes as Windows Server nodes shouldn't run the ingress controller. You specify a node selector by using the `--set nodeSelector` parameter to tell the Kubernetes scheduler to run the NGINX ingress controller only on Linux-based nodes.

   ```bash
   helm install nginx-ingress stable/nginx-ingress \
       --namespace ingress \
       --set controller.replicaCount=2 \
       --set controller.nodeSelector."beta\.kubernetes\.io/os"=linux \
       --set defaultBackend.nodeSelector."beta\.kubernetes\.io/os"=linux
   ```

3. After the installation is finished, you'll see an output similar to this example.

   ```output
   NAME: nginx-ingress
   LAST DEPLOYED: Mon Jan  6 15:18:42 2020
   NAMESPACE: ingress
   STATUS: deployed
   REVISION: 1
   TEST SUITE: None
   NOTES:
   The nginx-ingress controller has been installed.
   It may take a few minutes for the LoadBalancer IP to be available.
   You can watch the status by running 'kubectl --namespace ingress get services -o wide -w nginx-i
   ```

4. Next, let's check the public IP of the ingress service. It takes a few minutes for the service to acquire the public IP. Run the following command with a *watch* by adding the `-w` flag to see it update in real time. Select `Ctrl+C` to stop watching.

   ```bash
   kubectl get service nginx-ingress-controller --namespace ingress -w
   ```

   The service shows `EXTERNAL-IP` as `<pending>` for a while until it finally changes to an actual IP.

   ```output
   NAME                       TYPE           CLUSTER-IP    EXTERNAL-IP     PORT(S)
   nginx-ingress-controller   LoadBalancer   10.2.0.162    13.68.177.68    80:32018/TCP,443:30245/
   ```

   Make a note of that EXTERNAL-IP, for example, 13.68.177.68.

Reconfigure the ratings web service to use ClusterIP

There's no need to use a public IP for the service because we're going to expose the deployment through ingress. Here, you'll change the service to use `ClusterIP` instead of `LoadBalancer`.

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

   ```bash
   code ratings-web-service.yaml
   ```

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

   ```YAML
   apiVersion: v1
   kind: Service
   metadata:
     name: ratings-web
     namespace: ratingsapp
   spec:
     selector:
       app: ratings-web
     ports:
     - protocol: TCP
       port: 80
       targetPort: 8080
     type: ClusterIP
   ```

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

4. You can't update the value of `type` on a deployed service. You have to delete the service and re-create it with the changed configuration.

   Run the following command to delete the service.

   ```bash
   kubectl delete service \
       --namespace ratingsapp \
       ratings-web
   ```

   Then, run the following command to re-create the service.

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

Create an Ingress resource for the ratings web service

In order for your Kubernetes ingress controller to route requests to the ratings-web service, you will need an ingress resource. The Ingress resource is where you specify the configuration of the Ingress controller.

Each Ingress resource will contain one or more Ingress rules, which specify an optional host, a list of paths to evaluate in the request, and a backend to route the request to. These rules are evaluated to determine the route that each request should take.

Let's set up an Ingress resource with a route to the ratings-web service.

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

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

2. Paste the following text in the file.

   ```YAML
   apiVersion: networking.k8s.io/v1beta1
   kind: Ingress
   metadata:
     name: ratings-web-ingress
     annotations:
       kubernetes.io/ingress.class: nginx
   spec:
     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 your ingress that 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. In the next unit, you'll configure SSL/TLS on that host name.

   > **Note**
   >
   > In this example, you use nip.io, which is a free service that provides wildcard DNS. You can use alternatives such as xip.io or sslip.io. Alternatively, you can use your domain name and set up the proper DNS records.

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 and deploy the ingress route file in the `ratingsapp` namespace.

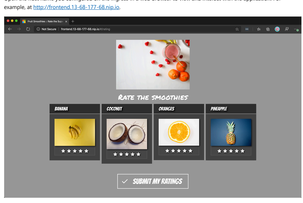
   ```bash
   kubectl apply \
       --namespace ratingsapp \
       -f ratings-web-ingress.yaml
   ```

   You'll see an output similar to this example.

   ```output
   ingress.networking.k8s.io/ratings-web-ingress created
   ```

Test the application

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



Summary

In this exercise, you deployed an NGINX Ingress controller and updated the **ratings-web** service to be accessible only from within the cluster. You then created an Ingress resource with a route to reverse proxy the deployment of the **ratings-web** service through a host name.

Next unit: Exercise - Enable SSL/TLS on the front-end ingress

Continue