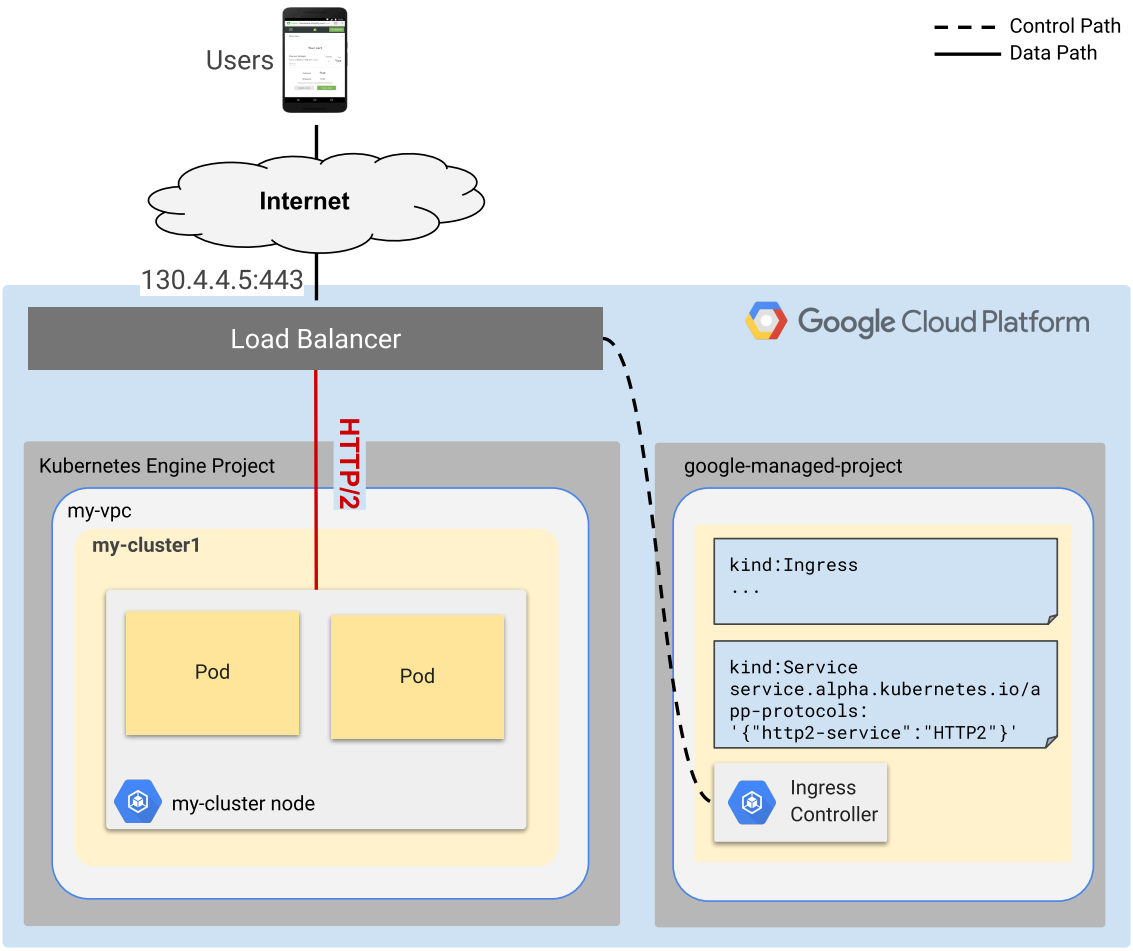
HTTP/2 for load balancing with Ingress

This page shows how to use Kubernetes Ingress (https://kubernetes.io/docs/concepts/services-networking/ingress/) and Service (https://kubernetes.io/docs/concepts/services-networking/service/) objects to configure an HTTP(S) load balancer (https://cloud.google.com/load-balancing/docs/https/) to use HTTP/2 (https://http2.github.io/) for communication with backend services. This feature is available starting with Google Kubernetes Engine version 1.11.2.

Overview

An HTTP(S) load balancer acts as a proxy between your clients and your application. Clients can use HTTP/1.1 or HTTP/2 to communicate with the load balancer proxy. However, the connection from the load balancer proxy to your application uses HTTP/1.1 by default. If your application, running in a Google Kubernetes Engine pod, is capable of receiving HTTP/2 requests, you configure the load balancer to use HTTP/2 when it forwards requests to your application.



(https://cloud.google.com/kubernetes-engine/images/ingress-http2.svg)

In this exercise, you create a Deployment, a Service, and an Ingress. You put a `cloud.google.com/app-protocols` annotation in your Service manifest to specify that the load balancer should use HTTP/2 to communicate with your application. Then you call your service and verify that your application received an HTTP/2 request.

Before you begin

To prepare for this task, perform the following steps:

- Ensure that you have enabled the Google Kubernetes Engine API.

  ENABLE GOOGLE KUBERNETES ENGINE API (HTTPS://CONSOLE.CLOUD.GOOGLE.COM/APIS/LIBRA

- Ensure that you have installed the Cloud SDK (https://cloud.google.com/sdk/downloads).

- Set your default project ID (https://support.google.com/cloud/answer/6158840):

```
gcloud config set project [PROJECT_ID]
```

- If you are working with zonal clusters, set your default compute zone (https://cloud.google.com/compute/docs/zones#available):

```
gcloud config set compute/zone [COMPUTE_ZONE]
```

- If you are working with regional clusters, set your default compute region (https://cloud.google.com/compute/docs/zones#available):

```
gcloud config set compute/region [COMPUTE_REGION]
```

- Update `gcloud` to the latest version:

```
gcloud components update
```

★ **Note:** You can override these default settings in `gcloud` commands using the `--project`, `--zone`, and `--region` operational flags.

- Read about the Kubernetes Ingress (https://kubernetes.io/docs/concepts/services-networking/ingress/) and Service (https://kubernetes.io/docs/concepts/services-networking/service/) resources.

Creating the Deployment

This Deployment manifest declares that you want to run two replicas of the `echoheaders` web application:

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: echoheaders
spec:
  replicas: 2
  template:
    metadata:
```

```
      labels:
        app: echoheaders
    spec:
      containers:
      - name: echoheaders
        image: k8s.gcr.io/echoserver:1.10
        ports:
        - containerPort: 8443
```

Copy the manifest to a file named `my-deployment.yaml`, and create the Deployment:

```
kubectl apply -f my-deployment.yaml
```

Creating the Service

Here's a manifest for the Service:

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    cloud.google.com/app-protocols: '{"my-port":"HTTP2"}'
  name: echoheaders
  labels:
    app: echoheaders
spec:
  type: NodePort
  ports:
  - port: 443
    targetPort: 8443
    protocol: TCP
    name: my-port
  selector:
    app: echoheaders
```

Save the manifest to a file named `my-service.yaml`, and create the Service:

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

View the Service:

```
kubectl get service echoheaders --output yaml
```

The output is similar to this:

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    cloud.google.com/app-protocols: '{"my-port":"HTTP2"}'
    ...
  labels:
    app: echoheaders
  name: echoheaders
  ...
spec:
  clusterIP: 10.39.251.148
  ...
  ports:
  - name: my-port
    nodePort: 30647
    port: 443
    protocol: TCP
    targetPort: 8443
  selector:
    app: echoheaders
  ...
  type: NodePort
...
```

For the purpose of this exercise, here are the important things to note about your Service:

- The Service has type `NodePort`. This type is required for Services that are going to be associated with an Ingress.
- Any Pod that has the label `app: echoheaders` is a member of the Service. The `selector` field specifies this.
- The Service has one port, and the port is named `my-port`. The `cloud.google.com/app-protocols` annotation specifies that `my-port` should use the HTTP/2 protocol.
- Traffic directed to the service on TCP port 443 is routed to TCP port 8443 in one of the member Pods. The `port` and `targetPort` fields specify this.

Creating the Ingress

Here's a manifest for the Ingress:

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: echomap
spec:
  backend:
    serviceName: echoheaders
    servicePort: 443
```

Copy the manifest to a file named `my-ingress.yaml`, and create the Ingress:

```
kubectl apply -f my-ingress.yaml
```

Wait a few minutes for the Kubernetes Ingress controller to configure an HTTP(S) load balancer, and then view the Ingress:

```
kubectl get ingress echomap --output yaml
```

The output is similar to this:

```
kind: Ingress
metadata:
  ...
  name: echomap
  ...
spec:
  backend:
    serviceName: echoheaders
    servicePort: 443
status:
  loadBalancer:
    ingress:
    - ip: 203.0.113.2
```

For the purpose of this exercise, here are the important things to note about your Ingress:

- The IP address for incoming traffic is listed under `loadBalancer:ingress`.

- Incoming requests are routed to a Pod that is a member of the `echoheaders` Service. In this exercise, the member pods have the label `app: echoheaders`.
- Requests are routed to the Pod on the target port specified in the `echoheaders` Service manifest. In this exercise, the Pod target port is 8443.

Verifying that your load balancer supports HTTP/2

GCLOUD CONSOLE

1. List your backend services:

```
gcloud compute backend-services list
```

2. Describe your backend service:

```
gcloud beta compute backend-services describe [BACKEND_SERVICE_NAME] --global
```

where [BACKEND_SERVICE_NAME] is the name of your backend service.

3. In the output, verify that the protocol is HTTP/2:

```
backends:
...
description: '{...,"kubernetes.io/service-port":"443","x-features":["HTTP2"]}
...
kind: compute#backendService
loadBalancingScheme: EXTERNAL
protocol: HTTP2
...
```

Calling your service

Wait a few minutes for the load balancer and backend service to be configured. Enter the external IP address of your load balancer in your browser's address bar.

The output shows information about the request from the load balancer to the Pod:

```
Hostname: echoheaders-7886d5bc68-xnrwj
...
Request Information:
    ...
    method=GET
    real path=/
    query=
    request_version=2
    request_scheme=https
    ...

Request Headers:
    ...
    x-forwarded-for=[YOUR_IP_ADDRESS], 203.0.113.2
    x-forwarded-proto=http
...
```

For the purpose of this exercise, here are the important things to note about the preceding output:

- The line `request_version=2` indicates that the request between the load balancer and the Pod used HTTP/2.
- The line `x-forwarded-proto=http` indicates that the request between you and the load balancer used HTTP 1.1, not HTTP/2.

What's next?

- Set up HTTP load balancing with Ingress (https://cloud.google.com/kubernetes-engine/docs/tutorials/http-balancer).
- Configure a static IP and domain name (https://cloud.google.com/kubernetes-engine/docs/tutorials/configuring-domain-name-static-ip) for your Ingress application using Ingress.
- Configure SSL certificates (https://cloud.google.com/kubernetes-engine/docs/how-to/ingress-multi-ssl) for your Ingress load balancer.
- If you have an application running on multiple Google Kubernetes Engine clusters in different regions, configure a multi-cluster Ingress

(https://cloud.google.com/kubernetes-engine/docs/how-to/multi-cluster-ingress) to route traffic to a cluster in the region closest to the user.

Last updated December 4, 2019.