Set up an external Application Load Balancer with Ingress

AUTOPILOT (/KUBERNETES-ENGINE/DOCS/CONCEPTS/AUTOPILOT-OVERVIEW)

STANDARD (/KUBERNETES-ENGINE/DOCS/CONCEPTS/TYPES-OF-CLUSTERS)

This tutorial shows how to run a web application behind an external Application Load Balancer (/load-balancing/docs/https) by configuring the Ingress (/kubernetes-engine/docs/concepts/ingress) resource.

Note: This process does not apply to an NGINX Ingress controller (https://github.com/kubernetes/ingress-nginx).

Background

Google Kubernetes Engine (GKE) offers integrated support for two types of Cloud Load Balancing for a publicly accessible application:

1. Ingress (/kubernetes-engine/docs/concepts/ingress)
2. External passthrough Network Load Balancer (/load-balancing/docs/network)

In this tutorial, you use Ingresses (/kubernetes-engine/docs/concepts/ingress).

Ingress

When you specify `kind: Ingress` in a resource manifest, you instruct GKE to create an Ingress (/kubernetes-engine/docs/concepts/ingress) resource. By including annotations and supporting workloads and Services, you can create a custom Ingress controller. Otherwise, GKE makes appropriate Google Cloud API calls to create an external Application Load Balancer (/load-balancing/docs/https). The load balancer's URL map's host rules and path matchers reference one or more backend services, where each backend service corresponds to a GKE Service (https://kubernetes.io/docs/concepts/services-networking/service/) of type `NodePort`, as referenced in the `Ingress`. The backends for each backend service are either instance groups or network endpoint groups (NEGs). NEGs are created when you configure container-native load balancing

(/kubernetes-engine/docs/how-to/container-native-load-balancing) as part of the configuration for your Ingress. For each backend service, GKE creates a Google Cloud health check, based on the readiness probe settings of the workload referenced by the corresponding GKE Service (https://kubernetes.io/docs/concepts/services-networking/service/).

If you are exposing an HTTP(S) service hosted on GKE, HTTP(S) load balancing (/compute/docs/load-balancing/http) is the recommended method for load balancing.

Note: The load balancers created by GKE are billed according to the regular Cloud Load Balancing pricing (/vpc/network-pricing#lb).

Note: To use Ingress, you must have the external Application Load Balancer add-on enabled. GKE clusters have external Application Load Balancers enabled by default; you must not disable it.

Note: Google Kubernetes Engine relies on a health check mechanism to determine the health status of the backend service. This mechanism cannot be used to perform Blackbox monitoring. To perform Blackbox monitoring, create an uptime check. For more information, see (Optional) Monitoring the availability and latency of your service (#monitor).

Objectives

- Create a GKE cluster.
- Deploy the sample web application to the cluster.
- Expose the sample app to the internet behind an external Application Load Balancer.

Costs

In this document, you use the following billable components of Google Cloud:

- GKE (/kubernetes-engine/pricing)

To generate a cost estimate based on your projected usage, use the pricing calculator (/products/calculator). New Google Cloud users might be eligible for a free trial (/free-trial).

When you finish the tasks that are described in this document, you can avoid continued billing by deleting the resources that you created. For more information, see Clean up (#clean-up).

Before you begin

Take the following steps to enable the Kubernetes Engine API:

1. Visit the Kubernetes Engine page (https://console.cloud.google.com/projectselector/kubernetes) in the Google Cloud console.
2. Create or select a project.
3. Wait for the API and related services to be enabled. This can take several minutes.
4. Make sure that billing is enabled for your Google Cloud project (/billing/docs/how-to/verify-billing-enabled#console).

Install the following command-line tools used in this tutorial:

- `gcloud` is used to create and delete Kubernetes Engine clusters. `gcloud` is included in the `gcloud` CLI (/sdk/docs/install).
- `kubectl` is used to manage Kubernetes, the cluster orchestration system used by Kubernetes Engine. You can install `kubectl` using `gcloud`:

  ```
  gcloud components install kubectl
  ```

Clone the sample code from GitHub:

```
git clone https://github.com/GoogleCloudPlatform/kubernetes-engine-samples
cd networking/kubernetes-engine-samples/load-balancing
```

Set defaults for the `gcloud` command-line tool

To save time typing your project ID (https://support.google.com/cloud/answer/6158840) and Compute Engine zone (/compute/docs/zones#available) options in the `gcloud` command-line tool, you can set the defaults:

```
gcloud config set project project-id
gcloud config set compute/zone compute-zone
```

Create a GKE cluster

Create a GKE Autopilot cluster:

```
gcloud container clusters create-auto loadbalancedcluster
```

Deploying a web application

The following manifest describes a Deployment (https://kubernetes.io/docs/concepts/workloads/controllers/deployment/) that runs the sample web application container image on an HTTP server on port 8080:

networking/load-balancing/web-deployment.yaml (https://github.com/GoogleCloudPlatform/kubernetes-engine-samples/blob/HEAD/networking/load-balancing/web-deployment.yaml)

udPlatform/kubernetes-engine-samples/blob/HEAD/networking/load-balancing/web-deployment.yaml)

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web
  namespace: default
spec:
  selector:
    matchLabels:
      run: web
  template:
    metadata:
      labels:
        run: web
    spec:
      containers:
      - image: us-docker.pkg.dev/google-samples/containers/gke/hello-app:1.0
        imagePullPolicy: IfNotPresent
        name: web
        ports:
        - containerPort: 8080
          protocol: TCP
```

Apply the resource to the cluster:

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

Exposing your Deployment inside your cluster

The following manifest describes a Service (https://kubernetes.io/docs/concepts/services-networking/service/) that makes the `web` deployment accessible within your container cluster:

networking/load-balancing/web-service.yaml (https://github.com/GoogleCloudPlatform/kubernetes-engine-samples/blob/HEAD/networking/load-balancing/web-service.yaml)

leCloudPlatform/kubernetes-engine-samples/blob/HEAD/networking/load-balancing/web-service.yaml)

```
apiVersion: v1
kind: Service
metadata:
  name: web
  namespace: default
spec:
  ports:
  - port: 8080
    protocol: TCP
    targetPort: 8080
  selector:
    run: web
  type: NodePort
```

1. Apply the resource to the cluster:

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

When you create a Service of type NodePort (https://kubernetes.io/docs/concepts/services-networking/service/#nodeport) with this command, GKE makes your Service available on a randomly selected high port number (e.g. 32640) on all the nodes in your cluster.

2. Verify that the Service is created and a node port is allocated:

```
kubectl get service web
```

Output:

```
NAME      TYPE       CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
web       NodePort   10.35.245.219   <none>        8080:32640/TCP   5m
```

In the sample output, the node port for the `web` Service is `32640`. Also, note that there is no external IP allocated for this Service. Since the GKE nodes are not externally accessible by default, creating this Service does not make your application accessible from the internet.

To make your HTTP(S) web server application publicly accessible, you must create an Ingress resource.

Creating an Ingress resource

Ingress (/kubernetes-engine/docs/concepts/ingress) is a Kubernetes resource that encapsulates a collection of rules and configuration for routing external HTTP(S) traffic to internal services.

On GKE, Ingress is implemented using Cloud Load Balancing (/load-balancing). When you create an Ingress in your cluster, GKE creates an HTTP(S) load balancer (/compute/docs/load-balancing/http) and configures it to route traffic to your application.

The following manifest describes an Ingress resource that directs traffic to your `web` Service:

networking/load-balancing/basic-ingress.yaml (https://github.com/GoogleCloudPlatform/kubernetes-engine-samples/blob/HEAD/networking/load-balancing/basic-ingress.yaml)

eCloudPlatform/kubernetes-engine-samples/blob/HEAD/networking/load-balancing/basic-ingress.yaml)

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: basic-ingress
spec:
  defaultBackend:
    service:
      name: web
      port:
        number: 8080
```

Apply the resource to the cluster:

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

After you deploy this manifest, Kubernetes creates an Ingress resource on your cluster. The GKE Ingress controller creates and configures an HTTP(S) Load Balancer (/compute/docs/load-balancing/http) according to the information in the Ingress, routing all external HTTP traffic (on port 80) to the `web` NodePort Service you exposed.

Note: To use Ingress, you must have the external Application Load Balancer add-on enabled. GKE clusters have external Application Load Balancers enabled by default; you must not disable it.

Visiting your application

Find out the external IP address of the load balancer serving your application by running:

```
kubectl get ingress basic-ingress
```

Output:

```
NAME            HOSTS     ADDRESS         PORTS     AGE
basic-ingress   *         203.0.113.12    80        2m
```

Note: It might take a few minutes for GKE to allocate an external IP address and set up forwarding rules before the load balancer is ready to serve your application. You might get errors such as HTTP 404 or HTTP 500 until the load balancer configuration is propagated across the globe.

Open the external IP address of your application in a browser and see a plain text HTTP response like the following:

```
Hello, world!
Version: 1.0.0
Hostname: web-6498765b79-fq5q5
```

You can visit Load Balancing (https://console.cloud.google.com/networking/loadbalancing) on the Google Cloud console and inspect the networking resources created by the GKE Ingress controller.

(Optional) Configuring a static IP address

When you expose a web server on a domain name, you need the external IP address of an application to be a static IP that does not change.

By default, GKE allocates ephemeral external IP addresses (/compute/docs/ip-addresses#ephemeraladdress) for HTTP applications exposed through an Ingress. Ephemeral addresses are subject to change. If you are planning to run your application for a long time, you must use a static external IP address (/compute/docs/ip-addresses/reserve-static-external-ip-address).

Note that after you configure a static IP for the Ingress resource, deleting the Ingress does not delete the static IP address associated with it. Make sure to clean up the static IP addresses you configured when you no longer plan to use them again.

Warning: The following instructions create a static IP address and then reference the IP address in an Ingress manifest. If you modify an existing Ingress to use a static IP address instead of an ephemeral IP address, GKE might change the IP address of the load balancer when GKE re-creates the forwarding rule of the load balancer.

To configure a static IP address, complete the following steps:

1. Reserve a static external IP address (/compute/docs/ip-addresses/reserve-static-external-ip-address) named **`web-static-ip`**:

   gcloudConfig Connector...
   (#gcloud)

   ```
   gcloud compute addresses create web-static-ip --global
   ```

2. The **`basic-ingress-static.yaml`** manifest adds an annotation on Ingress to use the static IP resource named **`web-static-ip`**:

   networking/load-balancing/basic-ingress-static.yaml (https://github.com/GoogleCloudPlatform/kubernetes-engine-samples/blob/HEAD/networking/load-balancing/basic-ingress-static.yaml)

   Platform/kubernetes-engine-samples/blob/HEAD/networking/load-balancing/basic-ingress-static.yaml)

   ```
   apiVersion: networking.k8s.io/v1
   kind: Ingress
   metadata:
     name: basic-ingress
     annotations:
       kubernetes.io/ingress.global-static-ip-name: "web-static-ip"
   spec:
     defaultBackend:
       service:
         name: web
         port:
           number: 8080
   ```

   View the manifest:

   ```
   cat basic-ingress-static.yaml
   ```

3. Apply the resource to the cluster:

```
kubectl apply -f basic-ingress-static.yaml
```

4. Check the external IP address:

```
kubectl get ingress basic-ingress
```

Wait until the IP address of your application changes to use the reserved IP address of the **web-static-ip** resource.

It might take a few minutes to update the existing Ingress resource, re-configure the load balancer, and propagate the load balancing rules across the globe. After this operation completes, GKE releases the ephemeral IP address previously allocated to your application.

Note: Unused static external IP address are billed according to the regular IP address billing (/compute/pricing#ipaddress).

(Optional) Serving multiple applications on a load balancer

You can run multiple services on a single load balancer and public IP by configuring routing rules on the Ingress. By hosting multiple services on the same Ingress, you can avoid creating additional load balancers (which are billable resources) for every Service that you expose to the internet.

The following manifest describes a Deployment with version `2.0` of the same web application:

networking/load-balancing/web-deployment-v2.yaml (https://github.com/GoogleCloudPlatform/kubernetes-engine-samples/blob/HEAD/networking/load-balancing/web-deployment-v2.yaml)

Platform/kubernetes-engine-samples/blob/HEAD/networking/load-balancing/web-deployment-v2.yaml)

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web2
  namespace: default
spec:
  selector:
    matchLabels:
      run: web2
  template:
    metadata:
      labels:
        run: web2
    spec:
      containers:
      - image: us-docker.pkg.dev/google-samples/containers/gke/hello-app:2.0
        imagePullPolicy: IfNotPresent
        name: web2
        ports:
        - containerPort: 8080
          protocol: TCP
```

Apply the resource to the cluster:

```
kubectl apply -f web-deployment-v2.yaml
```

The following manifest describes a Service that exposes **`web2`** internally to the cluster on a NodePort Service called **`web2`**:

networking/load-balancing/web-service-v2.yaml (https://github.com/GoogleCloudPlatform/kubernetes-engine-samples/blob/HEAD/networking/load-balancing/web-service-v2.yaml)

loudPlatform/kubernetes-engine-samples/blob/HEAD/networking/load-balancing/web-service-v2.yaml)

```
apiVersion: v1
kind: Service
metadata:
  name: web2
  namespace: default
spec:
  ports:
  - port: 8080
    protocol: TCP
    targetPort: 8080
  selector:
    run: web2
  type: NodePort
```

Apply the resource to the cluster:

```
kubectl apply -f web-service-v2.yaml
```

The following manifest describes an Ingress resource that:

- routes the requests with path starting with `/v2/` to the `web2` Service
- routes all other requests to the `web` Service

networking/load-balancing/fanout-ingress.yaml (https://github.com/GoogleCloudPlatform/kubernetes-engine-samples/blob/HEAD/networking/load-balancing/fanout-ingress.yaml)

CloudPlatform/kubernetes-engine-samples/blob/HEAD/networking/load-balancing/fanout-ingress.yaml)

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: fanout-ingress
spec:
  rules:
  - http:
      paths:
      - path: /*
        pathType: ImplementationSpecific
        backend:
          service:
            name: web
            port:
              number: 8080
      - path: /v2/*
        pathType: ImplementationSpecific
        backend:
          service:
            name: web2
            port:
              number: 8080
```

Apply the resource to the cluster:

```
kubectl create -f fanout-ingress.yaml
```

After the Ingress is deployed, run `kubectl get ingress fanout-ingress` to find out the public IP address of the cluster.

Note: It might take a few minutes for GKE to allocate an external IP address and prepare the load balancer. You might get errors like HTTP 404 and HTTP 500 until the load balancer is ready to serve the traffic.

Then visit the IP address to see that both applications are reachable on the same load balancer:

- Visit `http://<IP_ADDRESS>/` and note that the response contains `Version: 1.0.0` (as the request is routed to the `web` Service)
- Visit `http://<IP_ADDRESS>/v2/` and note that the response contains `Version: 2.0.0` (as the request is routed to the `web2` Service)

The only supported wildcard character for the `path` field of an Ingress is the `*` character. The `*` character must follow a forward slash (`/`) and must be the last character in the pattern. For example, `/*`, `/foo/*`, and `/foo/bar/*` are valid patterns, but `*`, `/foo/bar*`, and `/foo/*/bar` are not.

A more specific pattern takes precedence over a less specific pattern. If you have both `/foo/*` and `/foo/bar/*`, then `/foo/bar/bat` is taken to match `/foo/bar/*`.

For more information about path limitations and pattern matching, see the URL Maps documentation (/load-balancing/docs/url-map).

(Optional) Monitoring the availability and latency of your serv

Google Cloud Uptime checks (/monitoring/uptime-checks) perform blackbox monitoring of applications from the viewpoint of the user, determining latency and availability from multiple external IPs to the IP address of the load balancer. In comparison, Google Cloud health checks perform an internal check against the Pod IPs, determining availability at the instance level. The checks are complementary and provide a holistic picture of application health.

You can create an uptime check by using the Google Cloud console, the Cloud Monitoring API, or by using the Cloud Monitoring client libraries. For information, see Managing uptime checks (/monitoring/uptime-checks). If you want to create an uptime check by using the Google Cloud console, do the following:

1. Go to the **Services & Ingress** page in the Google Cloud console.

   Go to Services & Ingress (https://console.cloud.google.com/kubernetes/discovery)

2. Click the name of the Service you want to create an uptime check for.
3. Click **Create Uptime Check**.
4. In the **Create Uptime Check** pane, enter a title for the uptime check and then click **Next** to advance to the **Target** settings.

   The **Target** fields of the uptime check are automatically filled in using the information from the Service load balancer.

   For complete documentation on all the fields in an uptime check, see Creating an uptime check (/monitoring/uptime-checks#create).
5. Click **Next** to advance to the **Response Validation** settings.
6. Click **Next** to advance to the **Alert and Notification** section.

   To monitor an uptime check, you can create an alerting policy or view the uptime check dashboard. An alerting policy can notify you by email or through a different channel if your uptime check fails. For general information about alerting policies, see Introduction to alerting (/monitoring/alerts).

★ **Note:** You can create an alerting policy for an uptime check as part of the process of creating the uptime check. Creating an alerting policy is optional, but it is recommended. For information on how to create an alerting policy as an independent action, see Alerting on uptime checks (/monitoring/uptime-checks/uptime-alerting-policies)

7. Click **Create**.

Remarks

By default, Ingress performs a periodic health check by making a `GET` request on the `/` path to determine the health of the application, and expects a HTTP 200 response. If you want to check a different path or to expect a different response code, you can use a custom health check path (/kubernetes-engine/docs/concepts/ingress#health_checks).

Ingress supports more advanced use cases, such as:

- **Name-based virtual hosting:** You can use Ingress to reuse the load balancer for multiple domain names, subdomains, and to expose multiple Services on a single IP address and load balancer. Check out the simple fanout (https://kubernetes.io/docs/concepts/services-networking/ingress/#simple-fanout) and name-based virtual hosting (https://kubernetes.io/docs/concepts/services-networking/ingress/#name-based-virtual-hosting) examples to learn how to configure Ingress for these tasks.
- **HTTPS termination** (/compute/docs/load-balancing/http#tls_support): You can configure the Ingress (https://kubernetes.io/docs/concepts/services-networking/ingress/#tls) to terminate the HTTPS traffic using the Cloud Load Balancer.

Note: Always modify the properties of the Load Balancer using the Ingress object. Making changes directly on the load balancing resources might get lost or overridden by the GKE Ingress controller.

When an Ingress is deleted, the GKE Ingress controller cleans up the associated resources (except reserved static IP addresses) automatically.

Clean up

To avoid incurring charges to your Google Cloud account for the resources used in this tutorial, either delete the project that contains the resources, or keep the project and delete the individual resources.

1. **Delete any manually created forwarding rules and target proxies that reference the Ingress:**

   ★ **Note:** This step is needed only if the load balancer associated with the Ingress is updated manually using the Google Kubernetes Engine API or the Google Cloud console.

   A dangling target proxy that is referencing a GKE Ingress controller managed URL map will cause the deletion of the Ingress to fail in GKE versions 1.15.4-gke.22+. Inspect the Ingress resource to find an event with an error message similar to the following:

```
Error during GC: error running load balancer garbage collection routine:
```

In the preceding error message, `k8s2-um-tlw9rhgp-default-my82-target-proxy` is a manually created target https proxy that is still referencing the URL map `k8s2-um-tlw9rhgp-default-my-ingress-9ifnni82` which was created and managed by an Ingress controller.

These manually created frontend resources (both forwarding rule and target proxy) must be deleted before proceeding with the deletion of the Ingress.

2. **Delete the Ingress:** This step deallocates the ephemeral external IP address and the load balancing resources associated with your application:

```
kubectl delete ingress basic-ingress
```

If you followed the optional step to create an Ingress to route requests by path, then delete the Ingress:

```
kubectl delete ingress fanout-ingress
```

3. **Delete the static IP address:** Complete this step only if you followed the optional step to create a static IP address.

   - If you followed "Option 1" to convert an existing ephemeral IP address to static IP, then visit the Google Cloud console (https://console.cloud.google.com/networking/addresses/list) to delete the static IP address.
   - If you followed "Option 2" to create a new static IP address, then run the following command to delete the static IP address:

   ```
   gcloud compute addresses delete web-static-ip --global
   ```

4. **Delete the cluster:** This step deletes the compute nodes (/compute) of your container cluster and other resources such as the Deployments in the cluster:

```
gcloud container clusters delete loadbalancedcluster
```

What's next

- Check out the Ingress user guide (https://kubernetes.io/docs/concepts/services-networking/ingress/) for details about Ingress features.
- Configure static IP and a domain name (/kubernetes-engine/docs/tutorials/configuring-domain-name-static-ip) for your Ingress application using Ingress.
- Configure SSL certificates (/kubernetes-engine/docs/how-to/ingress-multi-ssl) for your Ingress load balancer.
- Learn about using Google-managed SSL certificates with Ingress (/kubernetes-engine/docs/how-to/managed-certs).
- If you have an application running on multiple Kubernetes Engine clusters in different regions, set up a multi-cluster Ingress (/kubernetes-engine/docs/how-to/multi-cluster-ingress) to route traffic to a cluster in the region closest to the user.
- Explore other Kubernetes Engine tutorials (/kubernetes-engine/docs/tutorials).
- Explore reference architectures, diagrams, and best practices about Google Cloud. Take a look at our Cloud Architecture Center (/architecture).

Last updated 2023-12-04 UTC.