

# NGINX Ingress Controller on GKE

In Kubernetes, [Ingress](#) allows external users and client applications access to HTTP services. Ingress consists of two components: an *Ingress Resource* and an *Ingress Controller*:

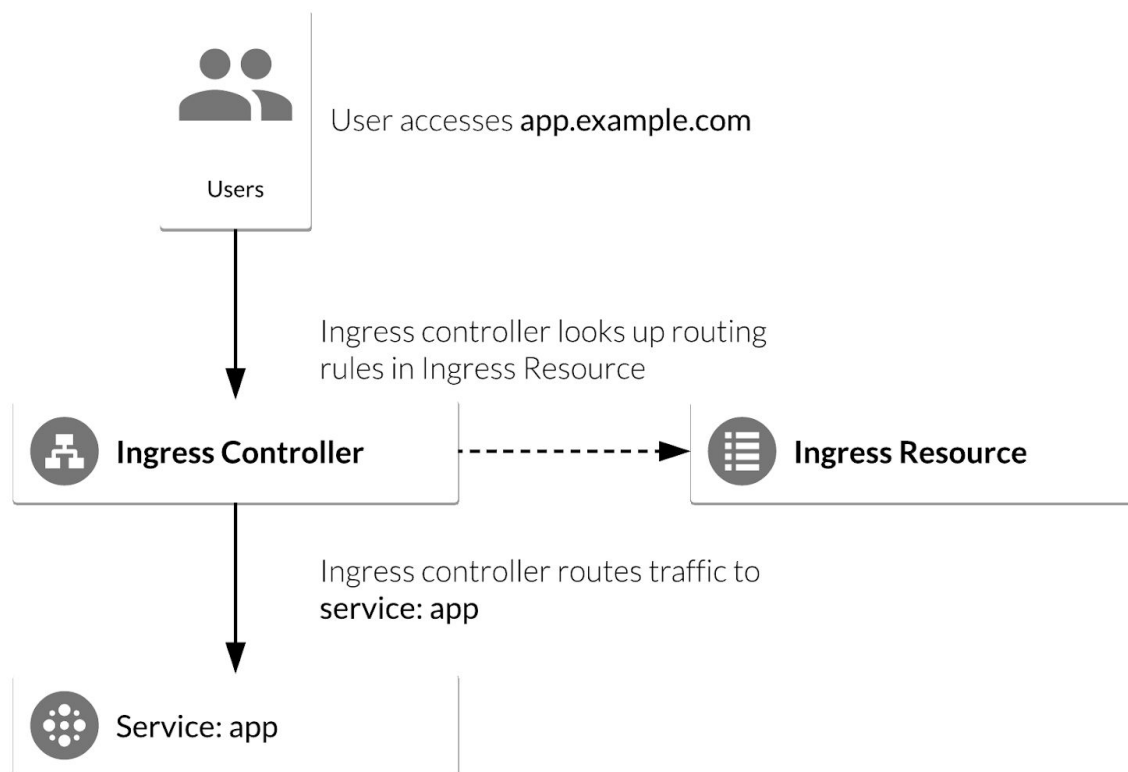
- **Ingress Resource** is a collection of rules for the inbound traffic to reach Services. These are Layer 7 (L7) rules that allow hostnames (and optionally paths) to be directed to specific Services in Kubernetes.
- **Ingress Controller** acts upon the rules set by the Ingress Resource, typically via an HTTP or L7 load balancer. It is vital that both pieces are properly configured so that traffic can be routed from an outside client to a Kubernetes Service.

NGINX—a high performance web server—is a popular choice for an Ingress Controller because of its robustness and the many features it boasts. For example, it supports:

- **Websockets**, which allows you to load balance Websocket applications.
- **SSL Services**, which allows you to load balance HTTPS applications.
- **Rewrites**, which allows you to rewrite the URI of a request before sending it to the application.
- **Session Persistence** (NGINX Plus only), which guarantees that all the requests from the same client are always passed to the same backend container.

- **JWTs** (NGINX Plus only), which allows NGINX Plus to authenticate requests by validating JSON Web Tokens (JWTs).

The following diagram illustrates the basic flow of an Ingress Controller in Google Cloud and gives you a rough idea of what you'll be building:



In this lab, you will configure a Kubernetes deployment with an Ingress Resource. You will use NGINX as an Ingress Controller, which you will use to route and load balance traffic from external clients to the deployment. More specifically, you will:

- Deploy a simple Kubernetes web application.
- Deploy an NGINX Ingress Controller using a stable Helm Chart.

- Deploy an Ingress Resource for the application that uses NGINX Ingress as the controller.
- Test NGINX Ingress functionality by accessing the Google Cloud L4 (TCP/UDP) Load Balancer frontend IP and ensure it can access the web application

- **Set a zone**

```
gcloud compute zones list
```

```
gcloud config set compute/zone us-central1-a
```

- **Create a Kubernetes cluster**

```
gcloud container clusters create nginx-tutorial --num-nodes 2
```

```
ahmed@osmi-contact@cloudshell: (digital-ci)$ gcloud container clusters create nginx-tutorial --num-nodes 2
WARNING: Currently VPC-native is not the default mode during cluster creation. In the future, this will become the default mode and can be disabled using '--no-enable-ip-alias' flag. Use '--[n
o]-enable-ip-alias' flag to suppress this warning.
WARNING: Newly created clusters and node-pools will have node auto-upgrade enabled by default. This can be disabled using the '--no-enable-autoupgrade' flag.
WARNING: Starting with version 1.18, clusters will have shielded GKE nodes by default.
WARNING: Your Pod address range ('--cluster-ip-v4-cidr') can accommodate at most 1008 node(s).
This will enable the autorepair feature for nodes. Please see https://cloud.google.com/kubernetes-engine/docs/node-auto-repair for more information on node autorepairs.
Creating cluster nginx-tutorial in us-central1-a... Cluster is being configured...
```

- **Deploy an application in Kubernetes Engine**

Now that you have Helm configured, let's deploy a simple web-based application from the Google Cloud Repository. This application will be used as the backend for the Ingress.

```
kubectl create deployment hello-app --image=gcr.io/google-samples/hello-app:1.0
kubectl expose deployment hello-app --port=8080
```

## 1- Deploy NGINX Ingress Controller

Let's go ahead and deploy the NGINX Ingress Controller. Run the following command to do so:

```
helm install nginx-ingress stable/nginx-ingress --set rbac.create=true
```

```
kubectl get svc
```

```
ahmedhosni_contact@cloudshell:~$ kubectl get services
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
cd-redis-master	ClusterIP	10.7.249.168	<none>	6379/TCP	7h44m
cd-spinnaker-halyard	ClusterIP	None	<none>	8064/TCP	7h44m
esp-srv	NodePort	10.7.245.216	<none>	80:30591/TCP	42d
hello-app	ClusterIP	10.7.248.154	<none>	8080/TCP	2m49s
incendiary-beetle-nginx-ingress-controller	LoadBalancer	10.7.249.41	104.154.77.34	80:31653/TCP,443:30746/TCP	42d
incendiary-beetle-nginx-ingress-default-backend	ClusterIP	10.7.253.119	<none>	80/TCP	42d
kubernetes	ClusterIP	10.7.240.1	<none>	443/TCP	42d
mycache-mcrouter	ClusterIP	None	<none>	5000/TCP	9d
mycache-memcached	ClusterIP	None	<none>	11211/TCP	9d
nginx-ingress-controller	LoadBalancer	10.7.245.101	34.123.127.207	80:30708/TCP,443:32503/TCP	102s
nginx-ingress-default-backend	ClusterIP	10.7.248.241	<none>	80/TCP	102s
spinnaker-1598656841-minio	ClusterIP	10.7.243.204	<none>	9000/TCP	7h14m
spinnaker-1598656841-redis-headless	ClusterIP	None	<none>	6379/TCP	7h14m
spinnaker-1598656841-redis-master	ClusterIP	10.7.253.208	<none>	6379/TCP	7h14m
spinnaker-1598656841-spi-halyard	ClusterIP	None	<none>	8064/TCP	7h14m

Note the second service, `nginx-ingress-default-backend`. The default backend is a Service which handles all URL paths and hosts the NGINX controller. The default backend exposes two URLs:

- `/healthz` that returns 200
- `/` that returns 404

Wait a few moments while the Google Cloud L4 Load Balancer gets deployed. Confirm that the `nginx-ingress-controller` Service has been deployed and that you have an external IP address associated with the service by running the following command:

```
kubectl get service nginx-ingress-controller
```

You receive a similar output:

```
ahmedhosni_contact@cloudshell:~$ kubectl get service nginx-ingress-controller
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
nginx-ingress-controller	LoadBalancer	10.7.245.101	34.123.127.207	80:30708/TCP,443:32503/TCP	5m56s

## 2- Configure Ingress Resource to use NGINX Ingress Controller

An Ingress Resource object is a collection of L7 rules for routing inbound traffic to Kubernetes Services.

Multiple rules can be defined in one Ingress Resource or they can be split up into multiple Ingress Resource manifests. The Ingress Resource also determines which controller to utilize to serve traffic.

This can be set with an annotation, `kubernetes.io/ingress.class`, in the metadata section of the Ingress Resource. For the NGINX controller, you will use the `nginx` value as shown below:

```
annotations: kubernetes.io/ingress.class: nginx
```

On Kubernetes Engine, if no annotation is defined under the metadata section, the Ingress Resource uses the Google Cloud GCLB L7 load balancer to serve traffic. This method can also be forced by setting the annotation's value to `gce`, like below:

```
annotations: kubernetes.io/ingress.class: gce
```

Let's create a simple Ingress Resource YAML file which uses the NGINX Ingress Controller and has one path rule defined by typing the following commands:

```
touch ingress-resource.yaml
nano ingress-resource.yaml
```

Add the following content in `ingress-resource.yaml` file:

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: ingress-resource
  annotations:
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/ssl-redirect: "false"
spec:
  rules:
  - http:
      paths:
      - path: /hello
        backend:
          serviceName: hello-app
          servicePort: 8080
```

then press **Ctrl-X**, then press **Y**, then press **Enter** to save the file.

The kind: Ingress dictates it is an Ingress Resource object. This Ingress Resource defines an inbound L7 rule for path /hello to service hello-app on port 8080.

Run the following command to apply those rules to our Kubernetes application:

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

Verify that Ingress Resource has been created:

```
ahmedhosni_contact@cloudshell:~$ kubectl apply -f ingress-resource.yaml
ingress.extensions/ingress-resource created
```

**Note:** The IP address for the Ingress Resource will not be defined right away. Wait a few moments for the ADDRESS field to get populate

Your output should resemble the following:

```
ahmedhosni_contact@cloudshell:~$ kubectl get ingress ingress-resource
NAME           HOSTS      ADDRESS      PORTS      AGE
ingress-resource *      34.68.80.96  80          2m6s
```

### 3- Test Ingress and default backend

You should now be able to access the web application by going to the EXTERNAL-IP/hello address of the **NGINX ingress controller** (found by running `kubectl get service nginx-ingress-controller`).

Open a new tab and go to the following, replacing the external-ip-of-ingress-controller with the external IP address of the NGINX ingress controller:

```
http://external-ip-of-ingress-controller/hello
```

Your page should look similar to the following:

```
← → ↻ 🏠 ⚠ Not secure | 34.122.155.117/hello
Apps ⚙️ Artifactory Logs - G... 🔄 Développons en Ja... 🔥

Hello, world!
Version: 1.0.0
Hostname: hello-app-6cc8c46757-z55nf
```

To check if the default-backend service is working properly, access any path (other than the path /hello defined in the Ingress Resource) and ensure you receive a 404 message. For example:

```
http://external-ip-of-ingress-controller/test
```

