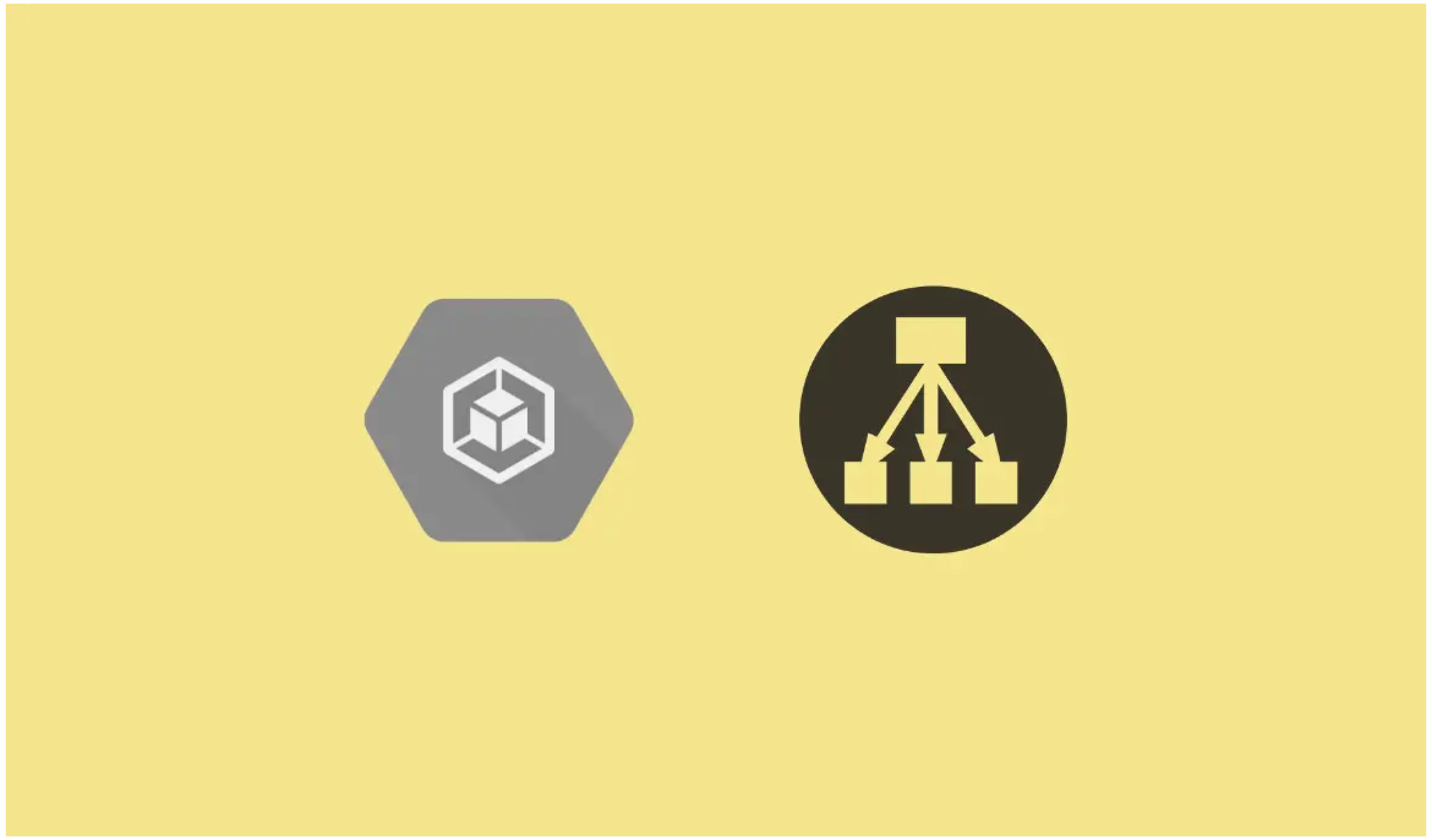
K — KUBERNETES

How to Setup Ingress on GKE using GKE Ingress Controller

by **devopscube** · June 24, 2021



This tutorial will guide you to setup Ingress on GKE using a GKE ingress controller that covers:

1. The need for ingress
2. The logic behind the GKE ingress controller
3. Creating your first ingress with a demo application.
4. Different ways of ingress domain/host and path mappings

Let's dive right in.

Exposing each service as a Loadbalancer is not an ideal solution to deal with Kubernetes ingress traffic.

You need a Kubernetes ingress controller to manage all the ingress traffic for the cluster. With direct DNS or a wildcard DNS mapping, you can route traffic to backend kubernetes services.

Also you can have multiple DNS attached to a single ingress Loadbalancer and route to different service backends using the Ingress controller.

Also, you can have path-based routing rules in the ingress resources to different kubernetes services.

The good thing is, GKE comes with an **inbuilt GKE ingress controller**. So you don't have to do any extra configurations to set up an ingress controller.

You can also set up other ingress controllers like the Nginx ingress controller. It depends upon the project requirements and the features supported by each controller implementation.

For this tutorial, we will just look at how to create an ingress object with the GKE ingress controllers.

How Does GKE Ingress Work?

As you probably know, for Kubernetes ingress to work, you need an Ingress controller.

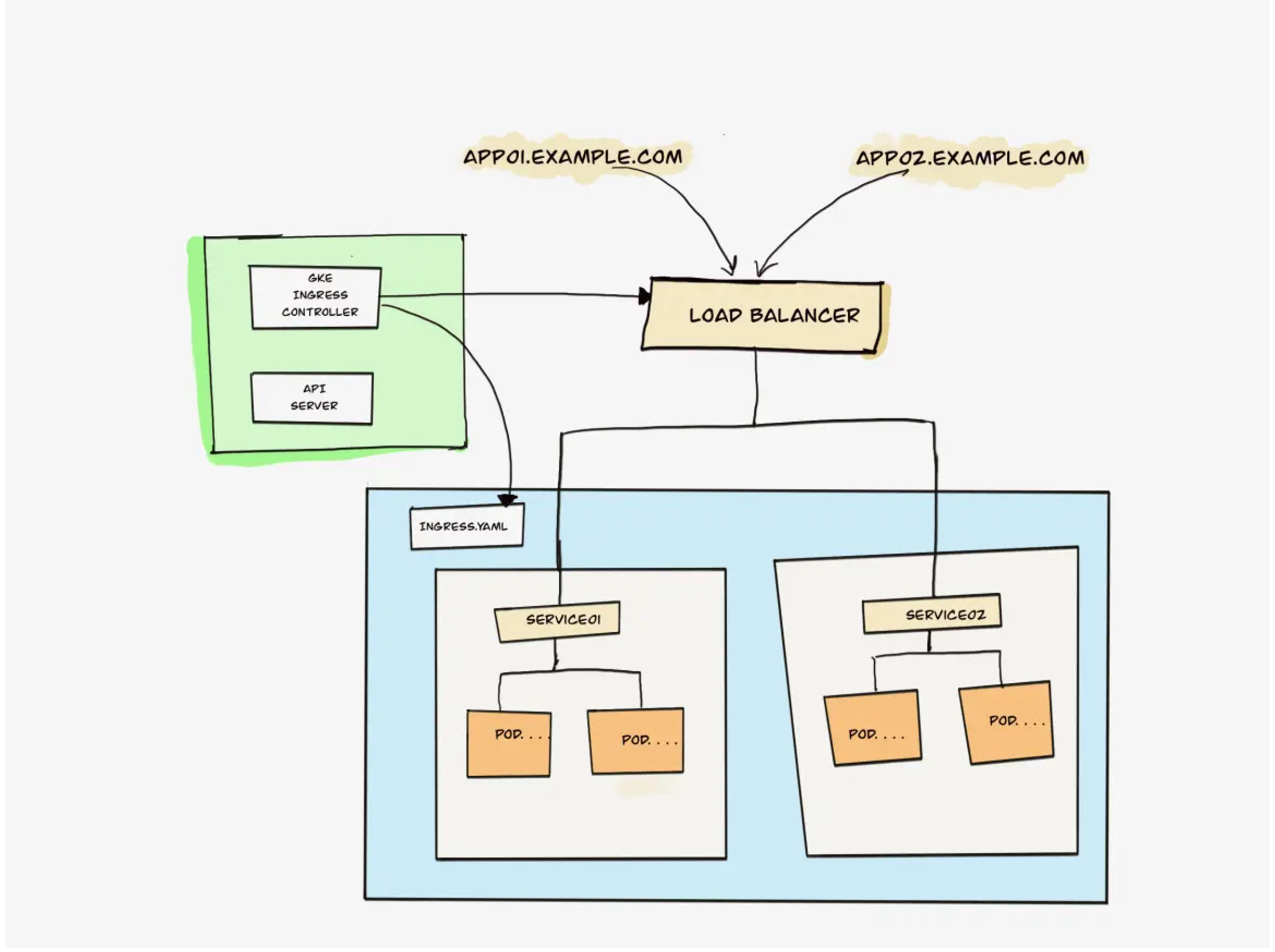
If you are not aware of ingress concepts, please read the ingress tutorial to understand how it works.

GKE has its own ingress controller called GKE ingress controller.

When you create an ingress object, GKE launches a Load Balancer (Public or Private) with all the routing rules mentioned in the ingress resource.

Since the Loadbalancer is outside the cluster, the backend service defined in the ingress resources should be of type LoadBalancer.

Whereas in normal ingress controller implementation like Nginx ingress controller, the proxy layer resides inside the cluster and it can talk to services without Nodeport.



In GKE, there is a concept of Network Endpoint Groups. Even though the backend service is of type NoePort, GKE doesn't randomly send traffic to any node in the cluster to reach the pods. With NEGs, all the traffic will be sent directly to the nodes where the pod resides.

Without NEGs, the traffic from the the Loadbalancer can read any node in the cluster and can result in extra network hops to reach the node where the pod resides.

Setup Ingress on GKE using a GKE Ingress Controller

Now, let's look at a practical example of setting up ingress using the GKE ingress controller.

To understand the GKE ingress workflow and configurations better, we will do the following.

1. Deploy an Nginx deployment with a `NodePort` service (Nodeport is a requirement)
2. Create an ingress object with static IP that has a rule to route traffic to the Nginx service endpoint.
3. Validate the ingress deployment by accessing Nginx over the Loadbalancer IP.

Lets get started with the setup.

Step 1: Deploy an Nginx deployment With Service Type NodePort

Save the following manifest as `nginx.yaml` . We are adding a custom index.html using a `configmap` which replaced the default Nginx `index.html` file.

> **Note:** For GKE ingress to work, the service type has to be NodePort. It is a requirement.

```
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  namespace: default
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 2
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:latest
        ports:
        - containerPort: 80
        readinessProbe:
          httpGet:
            scheme: HTTP
            path: /index.html
            port: 80
          initialDelaySeconds: 10
          periodSeconds: 5
        volumeMounts:
            - name: nginx-public
              mountPath: /usr/share/nginx/html/
      volumes:
      - name: nginx-public
        configMap:
```

```
kind: ConfigMap
```

```
metadata:
  name: nginx-index-html-configmap
  namespace: default
data:
  index.html: |
    <html>
    <h1>Welcome To Webapp 01</h1>
    </br>
    <h1>Hi! You are Trying to Access Webapp Through GKE Ingress </h1>
    </html
---
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
  namespace: default
spec:
  selector:
    app: nginx
  type: NodePort
  ports:
    - port: 80
      targetPort: 80
```

Create the deployment.

```
kubectl apply -f nginx.yaml
```

It gets deployed in the default namespace.

Validate the deployment. You should see two Replicas of Nginx.

```
kubectl get deployments
```

Validate the Nginx service endpoint. You should be abe able to see the randomly assigned NodePort.

```
kubectl get svc nginx-service
```

Step 2: Create Ingress for the Nginx Deployment

Now that we have a working deployment, we will create a ingress resource.

Lets start with a basic setup and then I will the address other options in the ingress.

What we are going to do is create an ingress that routes traffic to our Nginx service endpoint.

First, we need to create a Static Public IP address with the name `ingress-webapps` that we can use with the ingress. With static IP you can point the **required**

```
gcloud compute addresses create ingress-webapps --global
```

Save the following manifest as `ingress.yaml`

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: nginx-ingress
  annotations:
    kubernetes.io/ingress.class: "gce"
    kubernetes.io/ingress.global-static-ip-name: "ingress-webapps"
spec:
  rules:
  - http:
      paths:
      - path: "/*"
        pathType: ImplementationSpecific
        backend:
          service:
            name: nginx-service
            port:
              number: 80
```

As you can see we have two annotations in the manifest file.

1. `kubernetes.io/ingress.class` annotation with value `gce` tells GKE to create a public Load Balancer. If you don't specify it, it defaults to public only.
2. `kubernetes.io/ingress.global-static-ip-name` annotation with value `ingress-webapps` tells GKE to attach the static IP we created to the ingress Load Balancer.
3. We are creating a `http` rule to route all the traffic, ie `/*` to the `nginx-service` endpoint.

Lets create the ingress

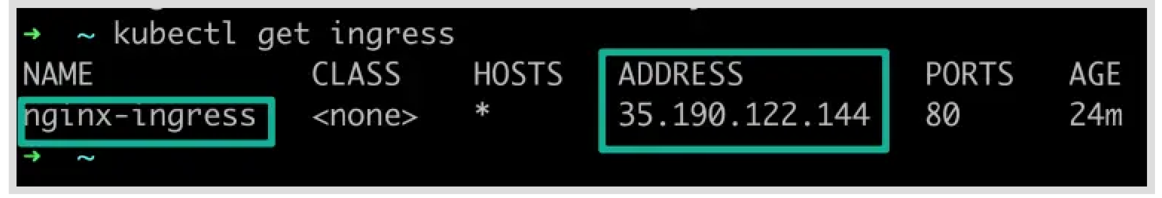
```
kubectl apply -f ingress.yaml
```

When you create ingress, it creates a Load balancer with the routing rules mentioned in the ingress YAML.

Step 3: Validate Ingress

Lets list the ingress and get the Load balancer IP.

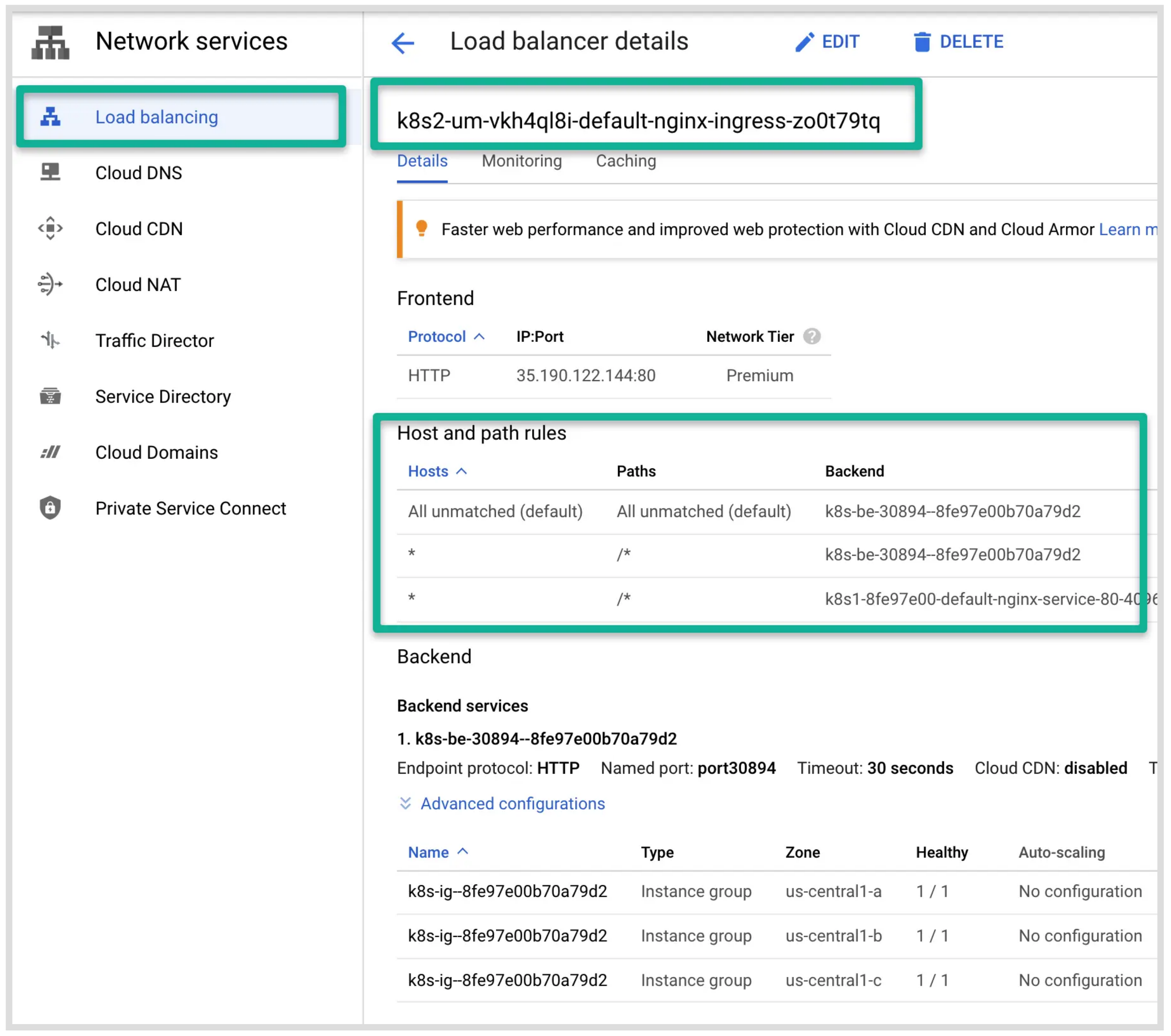
```
 kubectl get ingress
```



```
➜  ~ kubectl get ingress
NAME            CLASS    HOSTS   ADDRESS          PORTS   AGE
nginx-ingress   <none>   *       35.190.122.144   80      24m
➜  ~
```

After few minutes you should be able to access the nginx service running inside the cluster using the

Alternatively, you can check the Google Cloud load balancer dashboard to get more details about the Ingress Loadbalancer.



k8s2-um-vkh4ql8i-default-nginx-ingress-zo0t79tq

Faster web performance and improved web protection with Cloud CDN and Cloud Armor Learn m

Frontend

Protocol	IP:Port	Network Tier
HTTP	35.190.122.144:80	Premium

Host and path rules

Hosts	Paths	Backend
All unmatched (default)	All unmatched (default)	k8s-be-30894--8fe97e00b70a79d2
*	/*	k8s-be-30894--8fe97e00b70a79d2
*	/*	k8s1-8fe97e00-default-nginx-service-80-4096

Backend

Backend services

1. k8s-be-30894--8fe97e00b70a79d2

Endpoint protocol: **HTTP** Named port: **port30894** Timeout: **30 seconds** Cloud CDN: **disabled** T

Advanced configurations

Name	Type	Zone	Healthy	Auto-scaling
k8s-ig--8fe97e00b70a79d2	Instance group	us-central1-a	1 / 1	No configuration
k8s-ig--8fe97e00b70a79d2	Instance group	us-central1-b	1 / 1	No configuration
k8s-ig--8fe97e00b70a79d2	Instance group	us-central1-c	1 / 1	No configuration

Hosting Multiple Domains on Same GKE Ingress Loadbalancer

If you have a use case to host multiple applications with DNS on the same Load balancer, you can do it by mapping the same Loadbalancer IP in both the DNS A records.

Now, in the ingress specification, you need to add both the domain names with the respective backend service endpoints as shown below.

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: nginx-ingress
  annotations:
    kubernetes.io/ingress.class: "gce"
    kubernetes.io/ingress.global-static-ip-name: "ingress-webapps"
```

```
    host:  www.web01.com
    http:
      paths:
      - pathType: ImplementationSpecific
        path: "/"
        backend:
          service:
            name: web01-service
            port:
              number: 8080
  - host: "www.web02.com"
    http:
      paths:
      - pathType: ImplementationSpecific
        path: "/"
        backend:
          service:
            name: web02-service
            port:
              number: 8080
```

Path-Based Routing Using GKE Ingress

Now, let's say you have many backend services that need to be routed based on a specific path. In that case, your ingress would like the following.

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: nginx-ingress
  annotations:
    kubernetes.io/ingress.class: "gce"
    kubernetes.io/ingress.global-static-ip-name: "ingress-webapps"
spec:
  rules:
  - http:
      paths:
      - path: "/*"
        pathType: ImplementationSpecific
        backend:
          service:
            name: nginx-service
            port:
              number: 80
      - path: "/api"
        pathType: ImplementationSpecific
        backend:
          service:
            name: api-service
            port:
              number: 80
       - path: "/users"
        pathType: ImplementationSpecific
        backend:
          service:
            name: user-service
            port:
              number: 80
```

Multiple Ingress Resources with Single GKE Ingress Controller

resources mapped to a single ingress controller.

However, when it comes to GKE Load balancer based ingress controllers, you cannot map multiple ingress objects to a single GKE ingress controller.

For each ingress object or a resource, a separate Google Loadbalancer will be provisioned.

Here is what the google cloud document says about the multiple ingress resource mapping limitation.

> Combining multiple Ingress resources into a single Google Cloud load balancer is not supported.

The one way to circumvent this problem is to have all the ingress rules in a single ingress resource.

There is a limitation of a maximum of 50 paths per backend mapping. See out all the Quota limitations.

Customizing GKE Ingress Configurations

The example we have see uses all the default settings of GKE ingress controller.

But when it comes to production project requirements, you may have to tweak the configurations based on your application.

Few use cases are,

1. **Custom Headers:** You might have to add custom HTTP headers to the GKE ingress controller.
2. **Session Affinity:** If you want requests from a given user to be served by the same backend.

Refer to the GKE ingress controller features document to understand all the supported configurations.

To use the extra GKE ingress features, you need to deploy a Backend config with the required features. `Backendconfig` is a custom resource type available as part of GKE ingress controllers.

For example, to enable session affinity based on a cookie, you have to deploy the

```
apiVersion: cloud.google.com/v1
kind: BackendConfig
metadata:
  name: session-cookie-config
spec:
  sessionAffinity:
    affinityType: "GENERATED_COOKIE"
    affinityCookieTtlSec: 50
```

And for setting custom headers in GKE ingress, you have to use the following backend config.

```
apiVersion: cloud.google.com/v1
kind: BackendConfig
metadata:
  name: my-backendconfig
spec:
  customRequestHeaders:
    headers:
    - "X-Client-Region:{client_region}"
    - "X-Client-City:{client_city}"
    - "X-Client-CityLatLong:{client_city_lat_long}"
```

Conclusion

In this tutorial, we have learned to setup Ingress on GKE using a GKE ingress controller and Google cloud load balancer.

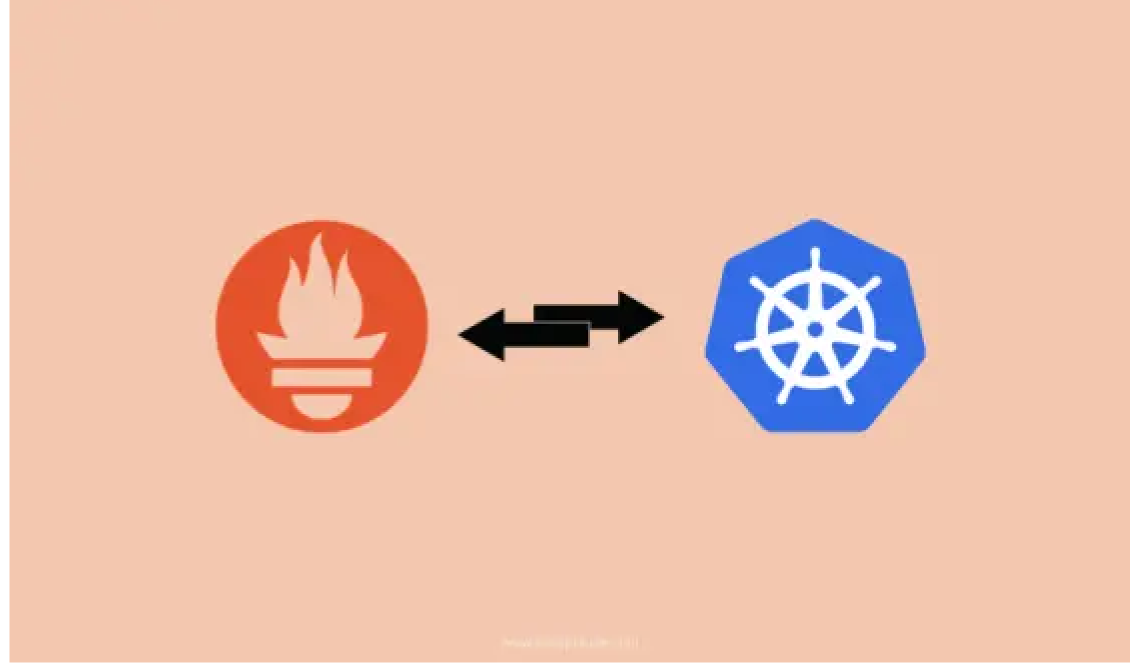
Choosing a GKE ingress controller vs. another ingress controller depends on the project requirements and features required in the ingress layer.

You might need more information or you may some some issues while setting it up.

Either way, drop a comment below.

devopscube

Established in 2014, a community for developers and system admins. Our goal is to continue to build a growing DevOps community offering the best in-depth articles, interviews, event listings, whitepapers, infographics and much more on DevOps.

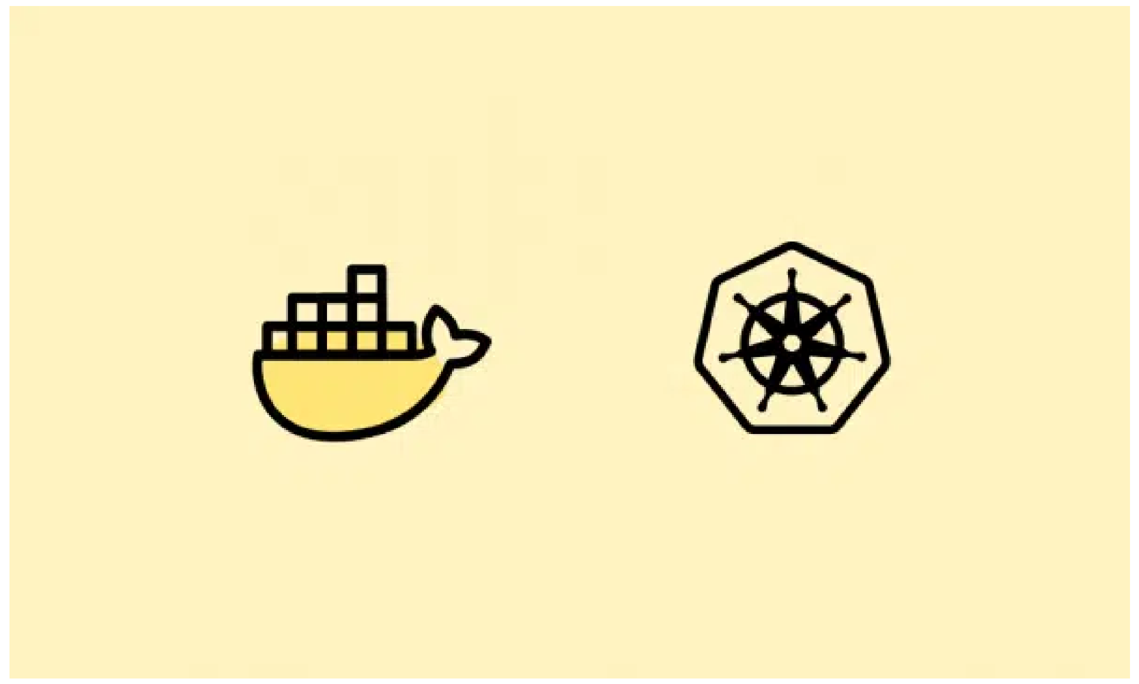


D — DEVOPS

How to Setup Prometheus Monitoring On Kubernetes Cluster

by **Bibin Wilson** · April 7, 2021

This article will guide you through setting up Prometheus on a Kubernetes cluster for monitoring the Kubernetes cluster....



K — KUBERNETES

How To Build Docker Image In Kubernetes Pod

by **devopscube** · July 24, 2021

This beginner's guide focuses on step by step process of setting up Docker image build in Kubernetes pod...

D — DEVOPS

Setting Up Alert Manager on Kubernetes – Beginners Guide

by **devopscube** · March 10, 2021

AlertManager is an open-source alerting system that works with the Prometheus Monitoring system. In the last article, I...

K — KUBERNETES

How To Setup Kube State Metrics on Kubernetes

by **Bibin Wilson** · November 3, 2019

In this blog we will look at what is Kube State Metrics and its setup on Kubernetes. What...

J — JENKINS

How to Setup Jenkins Build Agents on Kubernetes Cluster

by **Bibin Wilson** · July 4, 2021

In this Jenkins tutorial, I explained the detailed steps to set up Jenkins master and scale Jenkins build…

K — KUBERNETES

CKAD Exam Study Guide: A Complete Resource for CKAD Aspirants

by **Shishir Khandelwal** · June 28, 2021

This CKAD Exam study guide will help you prepare for the Certified Kubernetes Developer (CKAD) exam with all…