

Kubernetes Ingress

We have two options to get traffic outside the cluster

- NodePort
- LoadBalancer

Mostly the load balancer option is preferred in the public cloud providers.

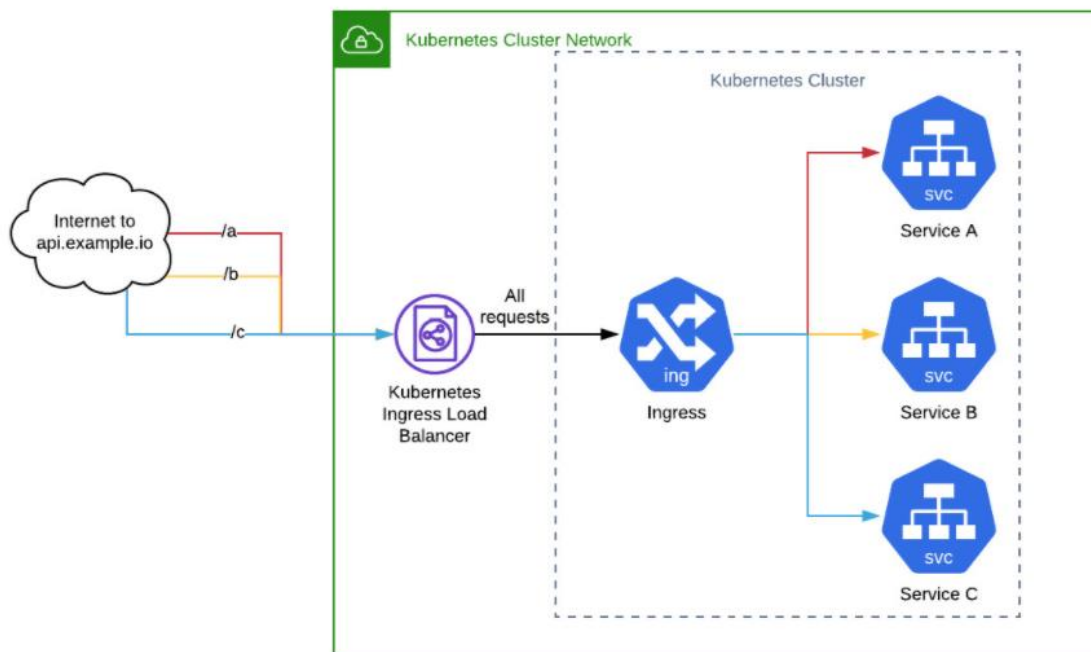
Limitations:

1. The problem is that one LoadBalancer can only point towards a single Kubernetes service object.

So now if you have 100 microservices -> you need 100 load balancers -> very expensive

2. Let's suppose that you have a web service running at **test.com** and you want **test.com/users** to go to one microservice and **test.com/notifications** to go to another completely separate microservice. Before the arrival of Ingress, you would need to set up your own Kubernetes Service and do the internal path resolution to your app.

The above issue is resolved using Kubernetes Ingress resource.



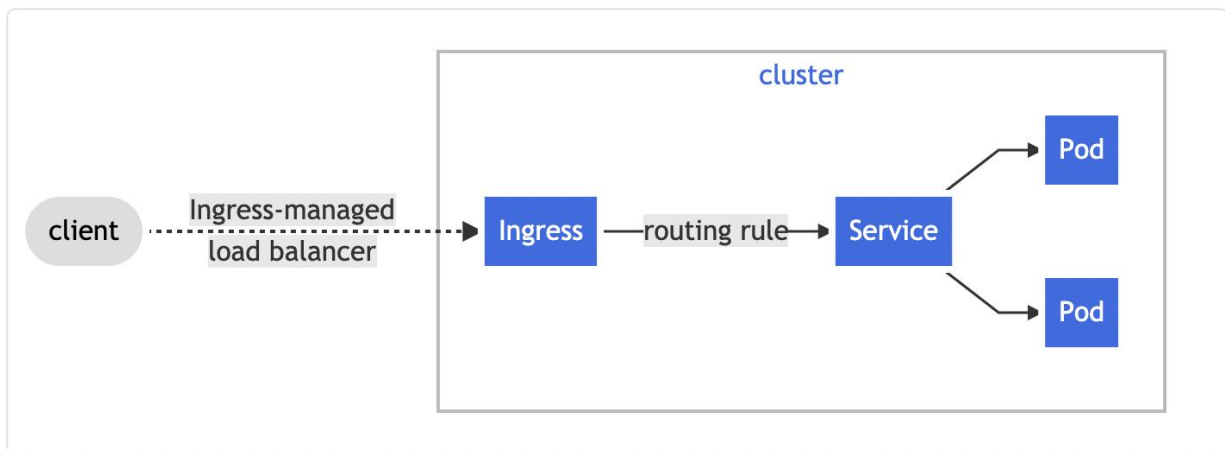
What is Ingress ?

An API object that manages external access to the services in a cluster, typically HTTP.

It also provides

- load balancing
- SSL termination
- name-based virtual hosting

Simple example where ingress sending all traffic to one service



Ingress Controller is responsible for fulfilling the ingress

What is Ingress Controller

In order for the Ingress resource to work, the cluster must have an ingress controller running. There are lot of ingress controllers available, you can find those details from their official [documentation](#).

Here, we will be using nginx ingress controller and do the deployment on GKE as per their [document](#)

Prerequisite:

1. Functional [Kubernetes Cluster](#) configured
2. Clone [this](#) git repo

Create an **ingress** controller

On GKE, we will run the below command which will create an ingress controller on our cluster

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/controller-v0.43.0/deploy/static/provider/cloud/deploy.yaml
```

```
nidhi@Nidhis-MacBook-Air ingress % kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/controller-v0.43.0/deploy/static/provider/cloud/deploy.yaml
namespace/ingress-nginx created
serviceaccount/ingress-nginx created
configmap/ingress-nginx-controller created
clusterrole.rbac.authorization.k8s.io/ingress-nginx created
clusterrolebinding.rbac.authorization.k8s.io/ingress-nginx created
role.rbac.authorization.k8s.io/ingress-nginx created
rolebinding.rbac.authorization.k8s.io/ingress-nginx created
service/ingress-nginx-controller-admission created
service/ingress-nginx-controller created
deployment.apps/ingress-nginx-controller created
validatingwebhookconfiguration.admissionregistration.k8s.io/ingress-nginx-admission created
serviceaccount/ingress-nginx-admission created
clusterrole.rbac.authorization.k8s.io/ingress-nginx-admission created
clusterrolebinding.rbac.authorization.k8s.io/ingress-nginx-admission created
role.rbac.authorization.k8s.io/ingress-nginx-admission created
rolebinding.rbac.authorization.k8s.io/ingress-nginx-admission created
job.batch/ingress-nginx-admission-create created
job.batch/ingress-nginx-admission-patch created
nidhi@Nidhis-MacBook-Air ingress %
```

```
kubectl get pods -n ingress-nginx
```

```
nidhi@Nidhis-MacBook-Air ingress % kubectl get pods -n ingress-nginx
NAME                                READY   STATUS    RESTARTS   AGE
ingress-nginx-admission-create-j5m5x 0/1     Completed 0           36s
ingress-nginx-admission-patch-zh88n  0/1     Completed 0           35s
ingress-nginx-controller-79b9595f96-n5mzw 1/1     Running   0           41s
```

```
kubectl get svc -n ingress-nginx
```

```
nidhi@Nidhis-MacBook-Air ingress % kubectl get svc -n ingress-nginx
NAME                                TYPE           CLUSTER-IP      EXTERNAL-IP      PORT(S)                                AGE
ingress-nginx-controller            LoadBalancer   10.35.240.122    34.67.167.77     80:32719/TCP,443:30800/TCP           2m10s
ingress-nginx-controller-admission ClusterIP       10.35.247.103    <none>           443/TCP                               2m10s
```

Basic Example of using Kubernetes Ingress Resource

Now we will run a sample application configured in this [yaml](#) file and expose it externally using Kubernetes Ingress Resource

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web
  namespace: default
spec:
  selector:
    matchLabels:
      run: web
  template:
    metadata:
      labels:
        run: web
    spec:
      containers:
        - image: gcr.io/google-samples/hello-app:1.0
          imagePullPolicy: IfNotPresent
          name: web
          ports:
            - containerPort: 8080
              protocol: TCP
---
apiVersion: v1
kind: Service
metadata:
  name: web
  namespace: default
spec:
  ports:
    - port: 8080
      protocol: TCP
      targetPort: 8080
  selector:
    run: web
  type: NodePort
```

Run the below command

```
cd ingress/basic-example
kubectl apply -f web.yaml
```

```
nidhi@Nidhis-MacBook-Air ingress % kubectl apply -f deployment.yaml
deployment.apps/web created
deployment.apps/web2 created
service/web created
service/web2 created
nidhi@Nidhis-MacBook-Air ingress %
```

Now we will deploy an ingress resource

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: basic-ingress
spec:
  backend:
    serviceName: web
    servicePort: 8080
```

Run the below command

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

```
nidhi@Nidhis-MacBook-Air ingress % kubectl apply -f fanout-ingress.yaml
ingress.networking.k8s.io/fanout-ingress created
nidhi@Nidhis-MacBook-Air ingress %
```

```
kubectl get ingress
```

```
nidhi@Nidhis-MacBook-Air ingress % kubectl get ingress
NAME           HOSTS    ADDRESS          PORTS    AGE
fanout-ingress *        35.192.14.67     80       88s
nidhi@Nidhis-MacBook-Air ingress %
```

Run the command again if IP didn't come

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.

Now you can open this IP on the browser



```
Hello, world!  
Version: 1.0.0  
Hostname: web-9bbd7b488-85nf2
```

Simple Fanout

A fanout configuration routes traffic from a single IP address to more than one Service, based on the HTTP URI being requested.

This yaml file will deploy 2 deployments(web and web1) and 2 services (web and web1)

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

```

    name: web2
    namespace: default
spec:
  selector:
    matchLabels:
      run: web2
  template:
    metadata:
      labels:
        run: web2
    spec:
      containers:
        - image: gcr.io/google-samples/hello-app:2.0
          imagePullPolicy: IfNotPresent
          name: web2
          ports:
            - containerPort: 8080
              protocol: TCP
---
apiVersion: v1
kind: Service
metadata:
  name: web
  namespace: default
spec:
  ports:
    - port: 8080
      protocol: TCP
      targetPort: 8080
  selector:
    run: web
  type: NodePort---
apiVersion: v1
kind: Service
metadata:
  name: web2
  namespace: default
spec:
  ports:
    - port: 8080
      protocol: TCP
      targetPort: 8080
  selector:
    run: web2
  type: NodePort

```

Run the below command

```

cd ingress/fanout-example
kubectl apply -f web.yaml

```

Now we will create an fanout ingress configuration

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: fanout-ingress
spec:
  rules:
  - http:
      paths:
      - path: /v1/*
        backend:
          serviceName: web
          servicePort: 8080
      - path: /v2/*
        backend:
          serviceName: web2
          servicePort: 8080
```

Run the command

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

Name-Based hosting

For name based hosting will be providing the hostname configuration in our yaml file

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: host-ingress
spec:
  rules:
  - host: "test.com"
    http:
      paths:
      - path: /test
        backend:
          serviceName: web
          servicePort: 8080
  - host: "abc.com"
    http:
      paths:
      - path: /abc
        backend:
          serviceName: web2
          servicePort: 8080
```

Run the below command


```
cd ingress/name-host
kubectl apply -f web.yaml
kubectl apply -f host-ingress.yaml
```

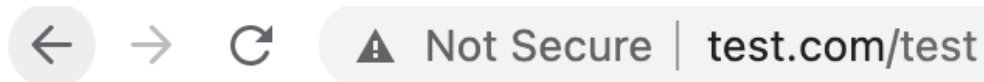
To access this url, edit /etc/hosts file on Mac

Add a line

```
IP of ingress    test.com
```

```
nidhi@Nidhis-MacBook-Air ingress % cat /private/etc/hosts
i##
# Host Database
#
# localhost is used to configure the loopback interface
# when the system is booting. Do not change this entry.
##
127.0.0.1        localhost
255.255.255.255 broadcasthost
34.67.167.77     test.com
::1              localhost
nidhi@Nidhis-MacBook-Air ingress %
```

Now try to access the browser as shown below:



```
Hello, world!
Version: 1.0.0
Hostname: web-9bbd7b488-85nf2
```

Hello, world!
Version: 2.0.0
Hostname: web2-74cf4946cc-qtn7s