**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.

**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:

name: nginx-index-html-configmap

---

apiVersion: v1

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

---

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

**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"

spec:

rules:

- 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

# nginx-default-ns.yaml

apiVersion: extensions/v1beta1

kind: Ingress

metadata:

name: nginx-ingress

namespace: default

annotations:

kubernetes.io/ingress.class: nginx

nginx.ingress.kubernetes.io/service-upstream: "true"

nginx.ingress.kubernetes.io/upstream-vhost: httpbin.default.svc.cluster.local

spec:

backend:

serviceName: httpbin

servicePort: 8000

---

# Deployment: nginx-ingress-controller

apiVersion: apps/v1

kind: Deployment

metadata:

name: nginx-ingress-controller

namespace: default

spec:

replicas: 1

selector:

matchLabels:

app: ingress-nginx

template:

metadata:

labels:

app: ingress-nginx

annotations:

prometheus.io/port: '10254'

prometheus.io/scrape: 'true'

# Do not redirect inbound traffic to Envoy.

traffic.sidecar.istio.io/includeInboundPorts: ""

traffic.sidecar.istio.io/excludeInboundPorts: "80,443"

# Exclude outbound traffic to kubernetes master from redirection.

traffic.sidecar.istio.io/excludeOutboundIPRanges: \_\_KUBE\_API\_SERVER\_IP\_\_

sidecar.istio.io/inject: 'true'

spec:

serviceAccountName: nginx-ingress-serviceaccount

containers:

- name: nginx

image: quay.io/kubernetes-ingress-controller/nginx-ingress-controller:0.12.0

securityContext:

runAsUser: 0

args:

- /nginx-ingress-controller

- --default-backend-service=$(POD\_NAMESPACE)/nginx-default-http-backend

- --configmap=$(POD\_NAMESPACE)/nginx-configuration

- --tcp-services-configmap=$(POD\_NAMESPACE)/nginx-tcp-services

- --udp-services-configmap=$(POD\_NAMESPACE)/nginx-udp-services

- --annotations-prefix=nginx.ingress.kubernetes.io

- --v=10

env:

- name: POD\_NAME

valueFrom:

fieldRef:

fieldPath: metadata.name

- name: POD\_NAMESPACE

valueFrom:

fieldRef:

fieldPath: metadata.namespace

ports:

- name: http

containerPort: 80

- name: https

containerPort: 443

livenessProbe:

failureThreshold: 8

initialDelaySeconds: 15

periodSeconds: 10

successThreshold: 1

timeoutSeconds: 1

httpGet:

path: /healthz

port: 10254

scheme: HTTP

readinessProbe:

failureThreshold: 8

periodSeconds: 10

successThreshold: 1

timeoutSeconds: 1

httpGet:

path: /healthz

port: 10254

scheme: HTTP

---

# Service: ingress-nginx

apiVersion: v1

kind: Service

metadata:

name: ingress-nginx

namespace: default

labels:

app: ingress-nginx

spec:

type: LoadBalancer

selector:

app: ingress-nginx

ports:

- name: http

port: 80

targetPort: http

- name: https

port: 443

targetPort: https

---

apiVersion: v1

kind: ConfigMap

metadata:

name: nginx-configuration

namespace: default

labels:

app: ingress-nginx

data:

ssl-redirect: "false"

---

apiVersion: v1

kind: ConfigMap

metadata:

name: nginx-tcp-services

namespace: default

labels:

app: ingress-nginx

---

apiVersion: v1

kind: ConfigMap

metadata:

name: nginx-udp-services

namespace: default

labels:

app: ingress-nginx

---

apiVersion: v1

kind: ServiceAccount

metadata:

name: nginx-ingress-serviceaccount

namespace: default

---

apiVersion: rbac.authorization.k8s.io/v1beta1

kind: ClusterRole

metadata:

name: nginx-ingress-clusterrole

namespace: default

rules:

- apiGroups:

- ""

resources:

- configmaps

- endpoints

- nodes

- pods

- secrets

verbs:

- list

- watch

- apiGroups:

- ""

resources:

- nodes

verbs:

- get

- apiGroups:

- ""

resources:

- services

verbs:

- get

- list

- watch

- apiGroups:

- "extensions"

resources:

- ingresses

verbs:

- get

- list

- watch

- apiGroups:

- ""

resources:

- events

verbs:

- create

- patch

- apiGroups:

- "extensions"

resources:

- ingresses/status

verbs:

- update

---

apiVersion: rbac.authorization.k8s.io/v1beta1

kind: Role

metadata:

name: nginx-ingress-role

namespace: default

rules:

- apiGroups:

- ""

resources:

- configmaps

- pods

- secrets

- namespaces

verbs:

- get

- apiGroups:

- ""

resources:

- configmaps

resourceNames:

# Defaults to "<election-id>-<ingress-class>"

# Here: "<ingress-controller-leader>-<nginx>"

# This has to be adapted if you change either parameter

# when launching the nginx-ingress-controller.

- "ingress-controller-leader-nginx"

verbs:

- get

- update

- apiGroups:

- ""

resources:

- configmaps

verbs:

- create

- apiGroups:

- ""

resources:

- endpoints

verbs:

- get

---

apiVersion: rbac.authorization.k8s.io/v1beta1

kind: RoleBinding

metadata:

name: nginx-ingress-role-nisa-binding

namespace: default

roleRef:

apiGroup: rbac.authorization.k8s.io

kind: Role

name: nginx-ingress-role

subjects:

- kind: ServiceAccount

name: nginx-ingress-serviceaccount

---

apiVersion: rbac.authorization.k8s.io/v1beta1

kind: ClusterRoleBinding

metadata:

name: nginx-ingress-clusterrole-nisa-binding

namespace: default

roleRef:

apiGroup: rbac.authorization.k8s.io

kind: ClusterRole

name: nginx-ingress-clusterrole

subjects:

- kind: ServiceAccount

name: nginx-ingress-serviceaccount

namespace: default

---

# Deployment: nginx-default-http-backend

apiVersion: apps/v1

kind: Deployment

metadata:

name: nginx-default-http-backend

namespace: default

labels:

app: nginx-default-http-backend

spec:

replicas: 1

selector:

matchLabels:

app: nginx-default-http-backend

template:

metadata:

labels:

app: nginx-default-http-backend

# rewrite kubelet's probe request to pilot agent to prevent health check failure under mtls

annotations: sidecar.istio.io/rewriteAppHTTPProbers: "true"

spec:

terminationGracePeriodSeconds: 60

containers:

- name: backend

# Any image is permissible as long as:

# 1. It serves a 404 page at /

# 2. It serves 200 on a /healthz endpoint

image: gcr.io/google\_containers/defaultbackend:1.4

securityContext:

runAsUser: 0

ports:

- name: http

containerPort: 8080

livenessProbe:

httpGet:

path: /healthz

port: 8080

scheme: HTTP

initialDelaySeconds: 30

timeoutSeconds: 5

resources:

limits:

cpu: 10m

memory: 20Mi

requests:

cpu: 10m

memory: 20Mi

---

# Service: nginx-default-http-backend

apiVersion: v1

kind: Service

metadata:

name: nginx-default-http-backend

namespace: default

labels:

app: nginx-default-http-backend

spec:

ports:

- name: http

port: 80

targetPort: http

selector:

app: nginx-default-http-backend

---

apiVersion: v1

kind: Namespace

metadata:

name: web

--- apiVersion: apps/v1

kind: Deployment

metadata:

name: web-server

namespace: web

spec:

selector:

matchLabels:

app: web

template:

metadata:

labels:

app: web

spec:

containers:

- name: httpd

image: httpd:2.4.53-alpine

ports:

- containerPort: 80

--- apiVersion: v1

kind: Service

metadata:

name: web-server-service

namespace: web

spec:

selector:

app: web

ports:

- protocol: TCP

port: 5000

targetPort: 80

--- apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

name: web-server-ingress

namespace: web

spec:

ingressClassName: nginx

rules:

- host: web.example.com

http:

paths:

- path: /

pathType: Prefix

backend:

service:

name: web-server-service

port:

number: 5000

--- apiVersion: v1

kind: Service

metadata:

name: example-load-balancer

spec:

selector:

app: web

ports:

- protocol: TCP

port: 80

targetPort: 80

type: LoadBalancer

------------------------------------------------------------------------------------------------------------------------------------------

**How to Create a Kubernetes Ingress Example setup.**

Let’s try to demonstrate ingress using path-based routing for 3 applications at a time where I used GKE to form a Kube cluster

Here deployment referred as Ingress-controller and configuration as Ingress-resource

\*\* so quick recap: Ingress is a decoupling layer between internet and nodePort where it opens the cluster to receive external traffic, defining the traffic routes to backend services results to ensuring reliable and secure communication \*\*

These are the steps we are going to performing to create the Kubernetes Ingress Path-based routing.

Deploy a web application

Expose your Deployment as a Service internally.

Create an Ingress resource.

Visit your applications.

Serving multiple applications on a Load Balancer.

**Step1: Deploying Web Application**

Create deployment yaml files kind: Deployment

**app1 – deployment yaml file**

for this app1 we are using apache4 image

apiVersion: extensions/v1beta1

kind: Deployment

metadata:

name: app1

namespace: default

spec:

selector:

matchLabels:

run: app1

template:

metadata:

labels:

run: app1

spec:

containers:

- image: punitporwal07/apache4ingress:1.0

imagePullPolicy: IfNotPresent

name: app1

ports:

- containerPort: 80

protocol: TCP

**app2 – deployment yaml file**

For this app2 we are using hello-app version 1 image from Google samples

apiVersion: extensions/v1beta1

kind: Deployment

metadata:

name: app2

namespace: default

spec:

selector:

matchLabels:

run: app2

template:

metadata:

labels:

run: app2

spec:

containers:

- image: gcr.io/google-samples/hello-app:1.0

imagePullPolicy: IfNotPresent

name: app2

ports:

- containerPort: 8080

protocol: TCP

**app3 – deployment yaml file**

For this app3 we are using hello-app version 2 image from Google samples

apiVersion: extensions/v1beta1

kind: Deployment

metadata:

name: app3

namespace: default

spec:

selector:

matchLabels:

run: app3

template:

metadata:

labels:

run: app3

spec:

containers:

- image: gcr.io/google-samples/hello-app:2.0

imagePullPolicy: IfNotPresent

name: app3

ports:

- containerPort: 8080

protocol: TCP

**Step2: Expose your Deployment as a Service internally.**

Exposing deployment as a service NodePort type kind : Service

**Service1 deployment YAML file**

this service is for the app1 with  apache container on port 80

apiVersion: v1

kind: Service

metadata:

name: app1

namespace: default

spec:

ports:

- port: 80

protocol: TCP

targetPort: 80

selector:

run: app1

type: NodePort

**Service2 deployment yaml file**

this service is for app2 with google sample app version1 container on port 8080

apiVersion: v1

kind: Service

metadata:

name: app2

namespace: default

spec:

ports:

- port: 8080

protocol: TCP

targetPort: 8080

selector:

run: app2

type: NodePort

**Service3 deployment yaml file**

this service is for app3 with google sample app version2 container on port 8080

apiVersion: v1

kind: Service

metadata:

name: app3

namespace: default

spec:

ports:

- port: 8080

protocol: TCP

targetPort: 8080

selector:

run: app3

type: NodePort

**Step3: Create an Ingress resource.**

Creating ingress resource with kind: Ingress

**Ingress resource yaml file**

you could see that we are defining a path based routing and mapping a backend service name and service port

for the path /  it goes to app1 service on port 80

for the path /v2 it goes app2 service on port 8080

for the path /v3 it goes app3 service on port 8080

apiVersion: extensions/v1beta1

kind: Ingress

metadata:

name: 3app-ingress

labels:

app: my-docker-apps

annotations:

spec:

rules:

- http:

paths:

- path: /

backend:

serviceName: app1

servicePort: 80

- path: /v2

backend:

serviceName: app2

servicePort: 8080

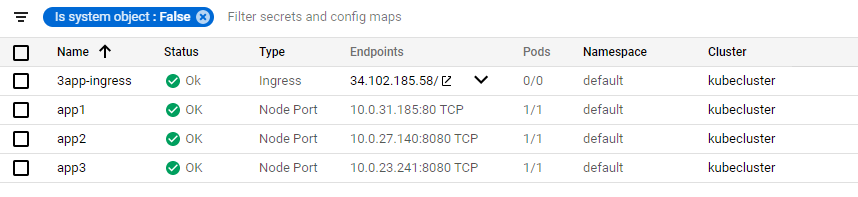
- path: /v3

backend:

serviceName: app3

servicePort: 8080

Ingress & services will look like on google cloud dashboard

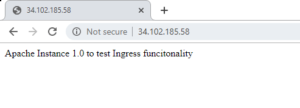
[](https://www.middlewareinventory.com/wp-content/uploads/2020/02/ing-services.png)

**Step4: Visit your applications.**

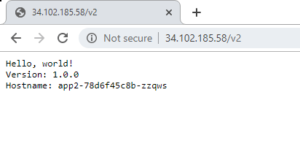
 Notice all three services share the Same IP, Unlike the Loadbalancer type where we need dedicated IP addresses.

And when you access ingress endpoint as defined in ingress definition.

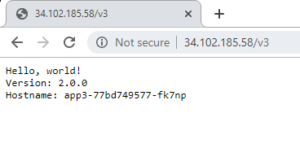
<http://34.102.185.58/>

[](https://www.middlewareinventory.com/wp-content/uploads/2020/02/app1.png)

<http://34.102.185.58/v2>

[](https://www.middlewareinventory.com/wp-content/uploads/2020/02/app2.png)

<http://34.102.185.58/v3>

[](https://www.middlewareinventory.com/wp-content/uploads/2020/02/app3.png)

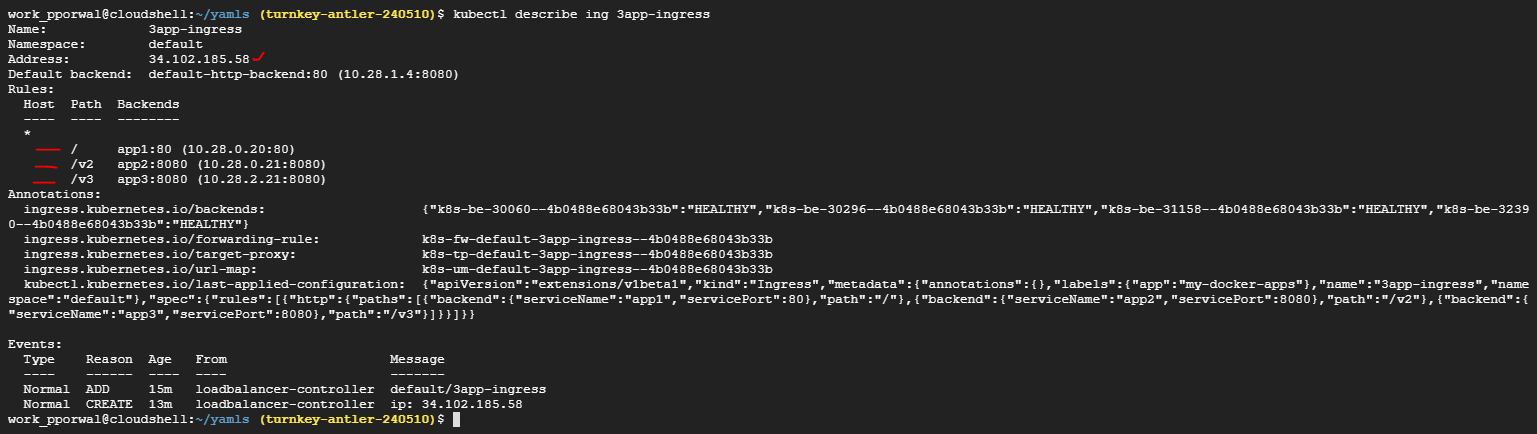
you will be able to access all three different application via a single endpoint, just changing context path as defined

**Step5: Serving multiple applications on a Load Balancer.**

The following command would show the internals of the created Kubernetes Ingress

$ kubectl describe ing 3app-ingress

you will see description of ingress definition

[](https://www.middlewareinventory.com/wp-content/uploads/2020/02/ing-desc.png)