LOAD BALANCER V/S INGRESS.

PROBLEM 1:

TRADITIONAL LOAD BALANCERS PROVIDES THE LOT OF ENTERPERISE LEVEL FEATURES LIKE

- Ratio Based Load Balancing

  Send 10 Requests to pod 1 10 requests to 5 pods

- Sticky Sessions:
  if one request is going to pod 1 all the request for that specific user go to that pod
- Path based Load balancing.
- Host Based Load Balancing
- Blacklisting.
- Whitelisting.

This are the Features that enterprise load balancer provides

BUT THIS FEATURES ARE NOT PROVIDED BY SERVICE TYPE LOAD BALANCER.

Load Balancer service in k8s provides simple round robin load balancer.

PROBLEM NO 2:

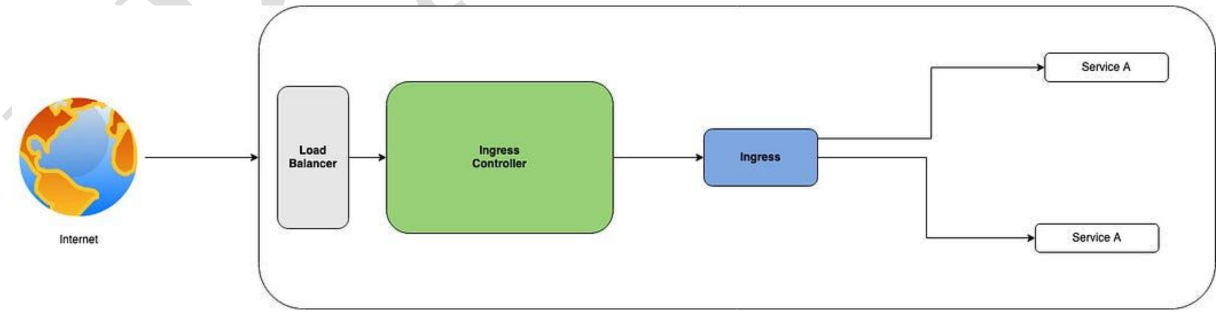
IF WE CAN EXPOSE THE APPLICATION LOAD BALANCER SERVICE MODE.

- SUPPOSE YOU HAVE 100 SERVICES FOR EACH SERVICE YOU NEED TO CREATE A LOAD BALANCER.
- SO U NEED TO CREATE 100 LOAD BALANCERS.
- FOR THESE 100 LOAD BALANCERS THE CLOUD PROVIDER CHARGES MONEY.

To Overcome This Problem we use Ingress in Kubernetes.

INGRESS:

An Ingress is a Kubernetes resource that allows traffic to come into your Kubernetes cluster. The external traffic could be via HTTP or HTTPS to a service running within your Kubernetes cluster. The service is exposed by your Ingress to allow inbound traffic.



Ingress provides externally available URLs, performs load balancing, terminates SSL/TLS, and offers name-based virtual hosting.

Ingress Controllers

An Ingress controller is a daemon running in a Pod that watches the */ingresses* endpoint on the API server. When a new endpoint is created, the daemon uses the configured set of rules to allow traffic into a service.

A controller uses Ingress Rules to handle traffic to and from outside the cluster.

There are many Ingress controllers Any tool capable of reverse proxying traffic should work.

As seen in the diagram above, this article focuses on NGINX Ingress controllers as NGINX-based controllers seem to be the most common. NGINX is a general-purpose implementation compatible with most Kubernetes deployments.

Why do we use Ingress because the load balancer supports the same thing?

- Ingress is used to manage the external traffic to the services within the cluster which provides features like host-based routing, path-based routing, SSL termination, and more.
- Where a Load balancer is used to manage the traffic but the load balancer does not provide the fine-grained access control like Ingress.

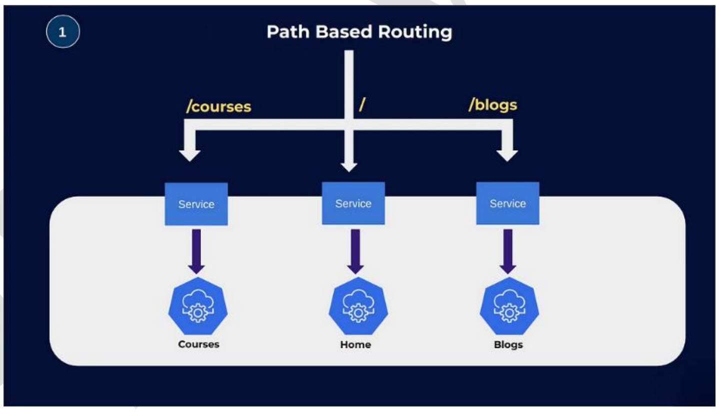
Example: Suppose you have multiple Kubernetes services running on your cluster and each service serves a different application such as example.com/app1 and example.com/app2. With the help of Ingress, you can achieve this. However, the Load Balancer routes the traffic based on the ports and can't handle the URL-based routing.

There are two types of Routing in Ingress:

Path-based routing:

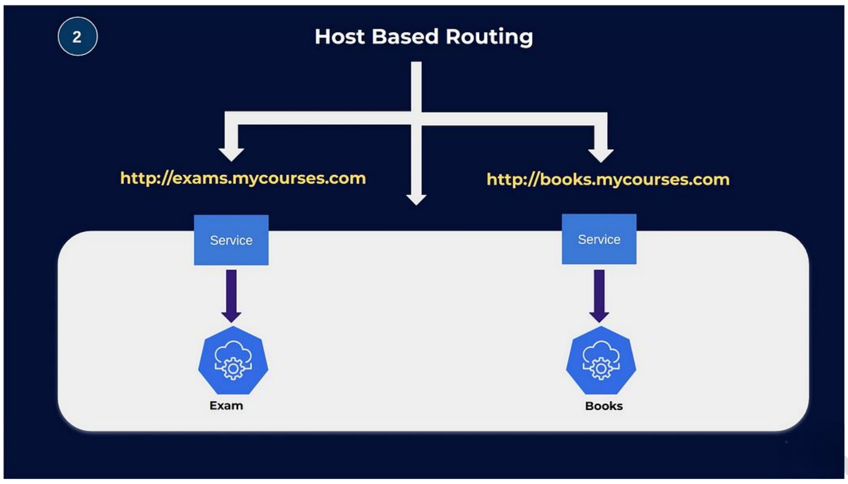
Path-based routing directs traffic to the different services based on the path.

Example: If you run an e-commerce site, traffic to yourdomain.com/courses might go to the shopping service, while traffic to yourdomain.com/blogs goes to the blog service



Host-based routing: Routes traffic based on the hostname.

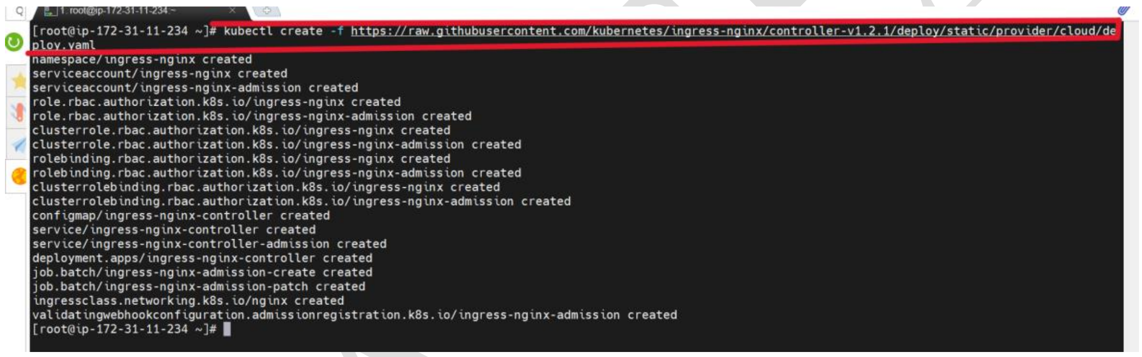
- **Example**: Traffic to https://exams.mycourses.com goes to your shopping service, while traffic to https://books.mycourses.com goes to your blog service.



To implement Ingress, we have to deploy Ingress Controllers. We can use any Ingress Controllers according to our requirements.

STEP 1: INSTALL INGRESS CONTROLLER.

kubectl create -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/controller-v1.2.1/deploy/static/provider/cloud/deploy.yaml

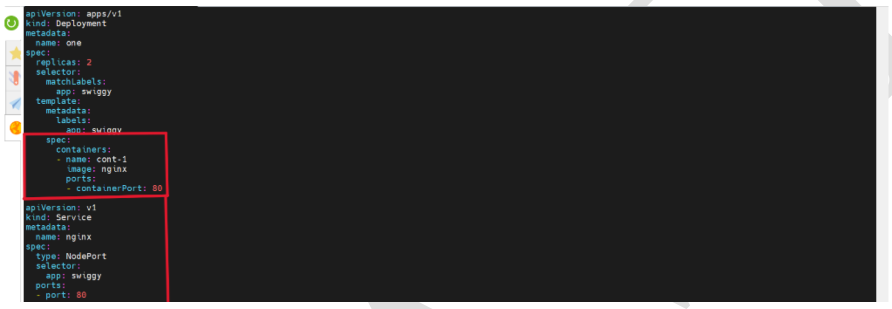


```
[root@ip-172-31-11-234 ~]# kubectl create -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/controller-v1.2.1/deploy/static/provider/cloud/de
ploy.yaml
namespace/ingress-nginx created
serviceaccount/ingress-nginx created
serviceaccount/ingress-nginx-admission created
role.rbac.authorization.k8s.io/ingress-nginx created
role.rbac.authorization.k8s.io/ingress-nginx-admission created
clusterrole.rbac.authorization.k8s.io/ingress-nginx created
clusterrole.rbac.authorization.k8s.io/ingress-nginx-admission created
rolebinding.rbac.authorization.k8s.io/ingress-nginx created
rolebinding.rbac.authorization.k8s.io/ingress-nginx-admission created
clusterrolebinding.rbac.authorization.k8s.io/ingress-nginx created
clusterrolebinding.rbac.authorization.k8s.io/ingress-nginx-admission created
configmap/ingress-nginx-controller created
service/ingress-nginx-controller created
service/ingress-nginx-controller-admission created
deployment.apps/ingress-nginx-controller created
job.batch/ingress-nginx-admission-create created
job.batch/ingress-nginx-admission-patch created
ingressclass.networking.k8s.io/nginx created
validatingwebhookconfiguration.admissionregistration.k8s.io/ingress-nginx-admission created
[root@ip-172-31-11-234 ~]#
```

STEP 2: CREATE A DEPLOYMENT AND SERVICE YAML FILE

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: one
spec:
  replicas: 2
  selector:
    matchLabels:
      app: swiggy
  template:
    metadata:
      labels:
        app: swiggy
    spec:
      containers:
      - name: cont-1
```

```
        image: nginx
        ports:
        - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
  name: nginx
spec:
  type: NodePort
  selector:
    app: swiggy
  ports:
  - port: 80
```



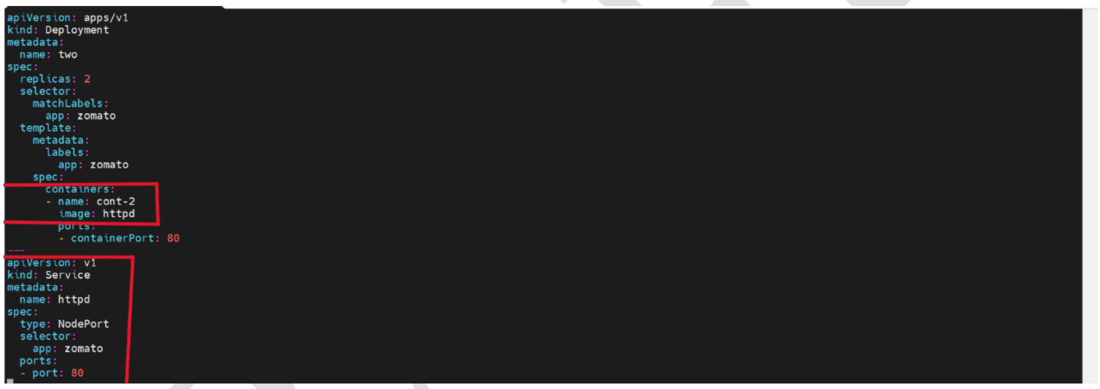
```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: one
spec:
  replicas: 2
  selector:
    matchLabels:
      app: swiggy
  template:
    metadata:
      labels:
        app: swiggy
    spec:
      containers:
      - name: cont-1
        image: nginx
        ports:
        - containerPort: 80
apiVersion: v1
kind: Service
metadata:
  name: nginx
spec:
  type: NodePort
  selector:
    app: swiggy
  ports:
  - port: 80
```

- This is deployment and service Manifest file releated to nginx application.

STEP 3: CREATE A SECOND DEPLOYMENT AND SERVICE YAML FILE.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: two
spec:
  replicas: 2
  selector:
    matchLabels:
      app: zomato
  template:
    metadata:
      labels:
        app: zomato
    spec:
```

```
    containers:
    - name: cont-2
      image: httpd
      ports:
      - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
  name: httpd
spec:
  type: NodePort
  selector:
    app: zomato
  ports:
  - port: 80
```



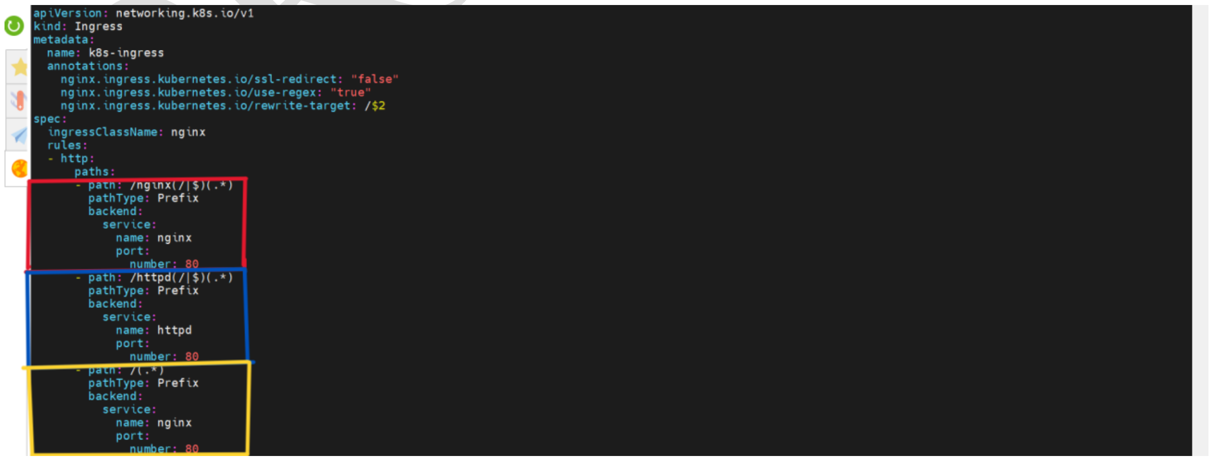
```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: two
spec:
  replicas: 2
  selector:
    matchLabels:
      app: zomato
  template:
    metadata:
      labels:
        app: zomato
    spec:
      containers:
      - name: cont-2
        image: httpd
        ports:
        - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
  name: httpd
spec:
  type: NodePort
  selector:
    app: zomato
  ports:
  - port: 80
```

- This is deployment and service Manifest file releated to httpd application.

STEP 4: CREATE A INGRESS YAML FILE.

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: k8s-ingress
  annotations:
    nginx.ingress.kubernetes.io/ssl-redirect: "false"
    nginx.ingress.kubernetes.io/use-regex: "true"
    nginx.ingress.kubernetes.io/rewrite-target: /$2
spec:
  ingressClassName: nginx
```

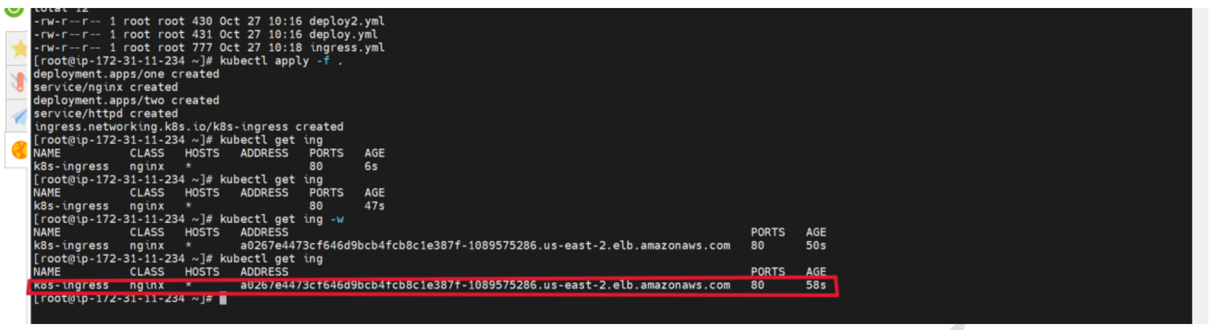
```
rules:
- http:
  paths:
  - path: /nginx(/|$)(.*)
    pathType: Prefix
    backend:
      service:
        name: nginx
        port:
          number: 80
  - path: /httpd(/|$)(.*)
    pathType: Prefix
    backend:
      service:
        name: httpd
        port:
          number: 80
  - path: /(.*)
    pathType: Prefix
    backend:
      service:
        name: nginx
        port:
          number: 80
```



```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: k8s-ingress
  annotations:
    nginx.ingress.kubernetes.io/ssl-redirect: "false"
    nginx.ingress.kubernetes.io/use-regex: "true"
    nginx.ingress.kubernetes.io/rewrite-target: /$2
spec:
  ingressClassName: nginx
  rules:
  - http:
      paths:
      - path: /nginx(/|$)(.*)
        pathType: Prefix
        backend:
          service:
            name: nginx
            port:
              number: 80
      - path: /httpd(/|$)(.*)
        pathType: Prefix
        backend:
          service:
            name: httpd
            port:
              number: 80
      - path: /(.*)
        pathType: Prefix
        backend:
          service:
            name: nginx
            port:
              number: 80
```

- If we the /(.*) path Requests that do not start with /nginx or /httpd will be routed to the nginx service on port 80
- If we hit the /http path the request goes to httpd application

- If we hit the /nginx path the request goes to nginx application

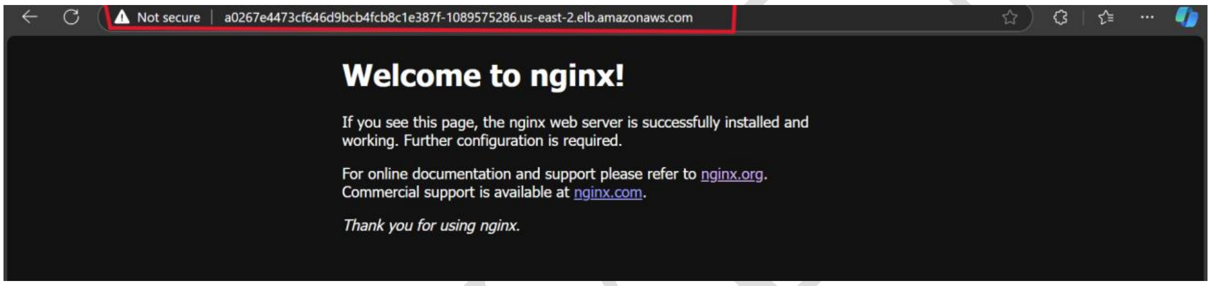


```
total 12
-rw-r--r-- 1 root root 430 Oct 27 10:16 deploy2.yml
-rw-r--r-- 1 root root 431 Oct 27 10:16 deploy.yml
-rw-r--r-- 1 root root 777 Oct 27 10:18 ingress.yml
[root@ip-172-31-11-234 ~]# kubectl apply -f .
deployment.apps/one created
service/nginx created
deployment.apps/two created
service/httpd created
ingress.networking.k8s.io/k8s-ingress created
[root@ip-172-31-11-234 ~]# kubectl get ing
NAME          CLASS   HOSTS   ADDRESS   PORTS   AGE
k8s-ingress   nginx   *                 80      6s
[root@ip-172-31-11-234 ~]# kubectl get ing
NAME          CLASS   HOSTS   ADDRESS   PORTS   AGE
k8s-ingress   nginx   *                 80      47s
[root@ip-172-31-11-234 ~]# kubectl get ing -w
NAME          CLASS   HOSTS   ADDRESS                                                                   PORTS   AGE
k8s-ingress   nginx   *       a0267e4473cf646d9bcb4fcb8c1e387f-1089575286.us-east-2.elb.amazonaws.com   80      50s
[root@ip-172-31-11-234 ~]# kubectl get ing
NAME          CLASS   HOSTS   ADDRESS                                                                   PORTS   AGE
k8s-ingress   nginx   *       a0267e4473cf646d9bcb4fcb8c1e387f-1089575286.us-east-2.elb.amazonaws.com   80      58s
[root@ip-172-31-11-234 ~]#
```

STEP 5: ACCESS OUR APPLICATION ON THE BASED PATH.

Access the application using load balancer .

If we don't mention any path in load balancer DNS we can access by default nginx application



Not secure | a0267e4473cf646d9bcb4fcb8c1e387f-1089575286.us-east-2.elb.amazonaws.com

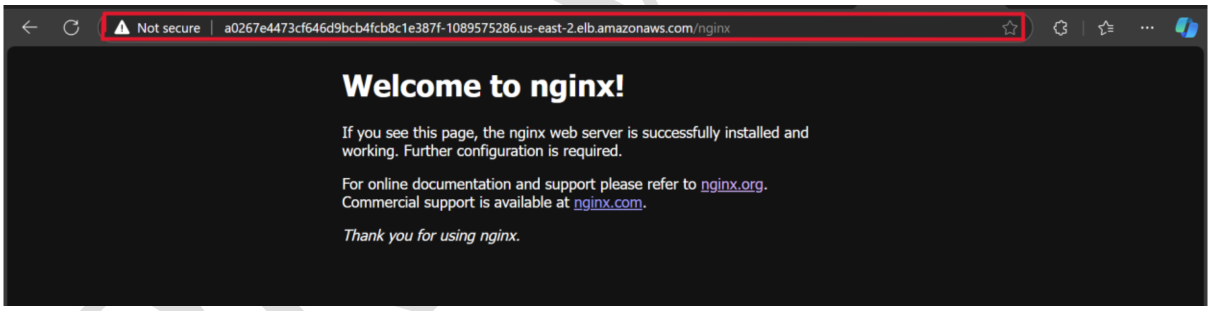
Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.

On /nginx we can access the nginx application:



Not secure | a0267e4473cf646d9bcb4fcb8c1e387f-1089575286.us-east-2.elb.amazonaws.com/nginx

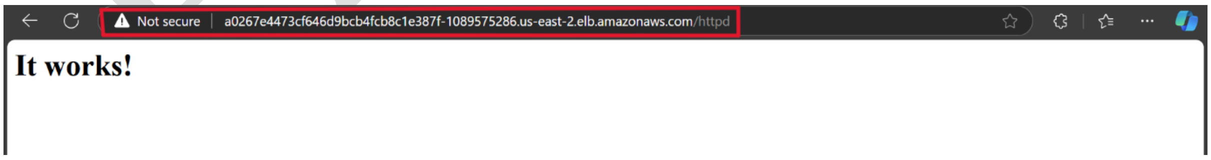
Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.

On /httpd we can access the httpd application.



Not secure | a0267e4473cf646d9bcb4fcb8c1e387f-1089575286.us-east-2.elb.amazonaws.com/httpd

It works!