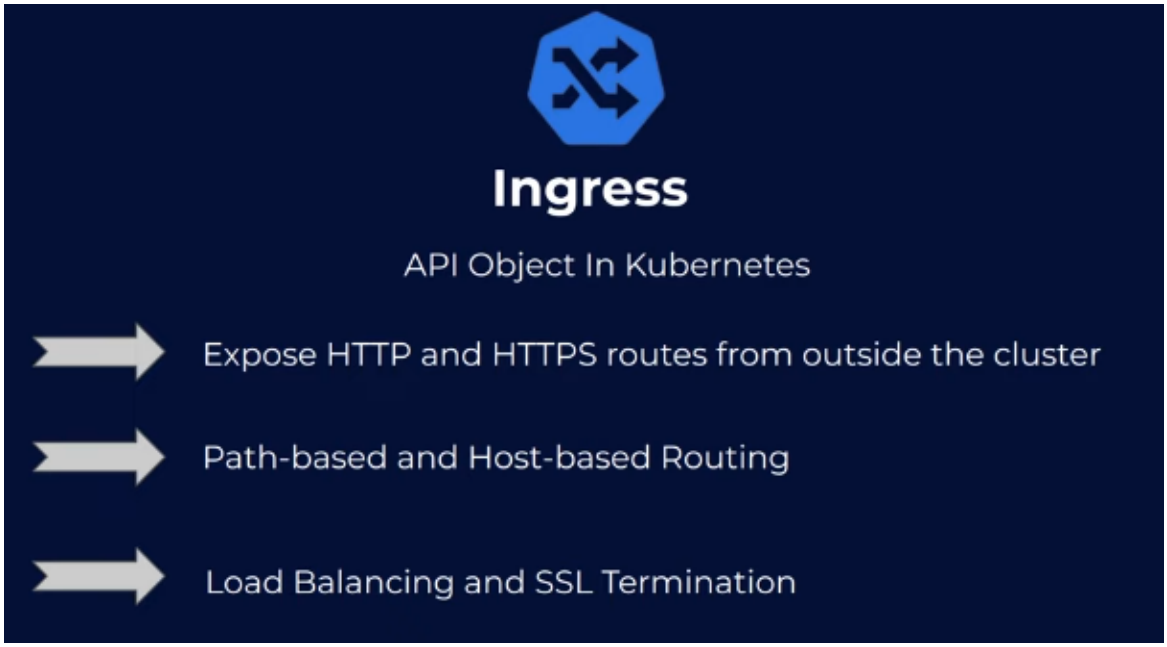
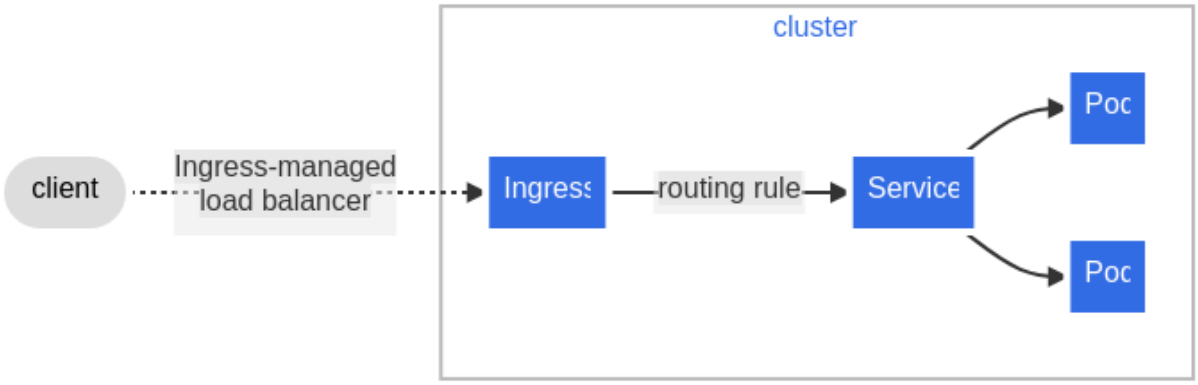
Nginx ingress Controller :

Ingress

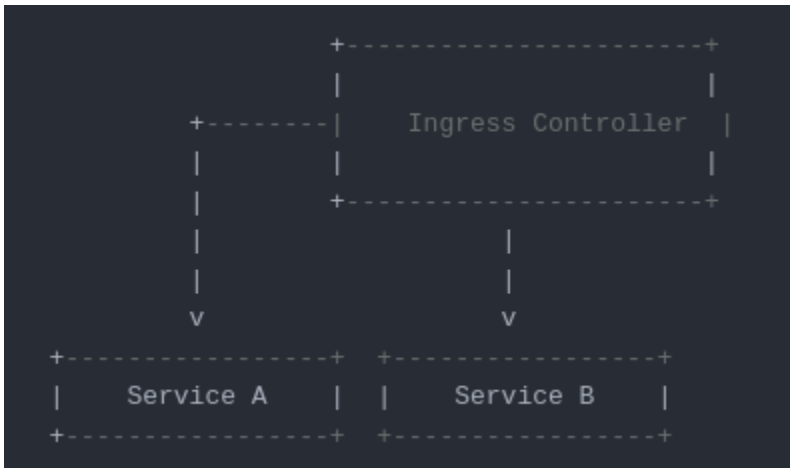


What is Ingress?



The concept of Ingress in Kubernetes allows you to configure external access to your services running in the cluster. An Ingress acts as a gateway or entry point that manages external traffic and routes it to the appropriate services based on defined rules.

Here is a diagram to help visualize the concept:



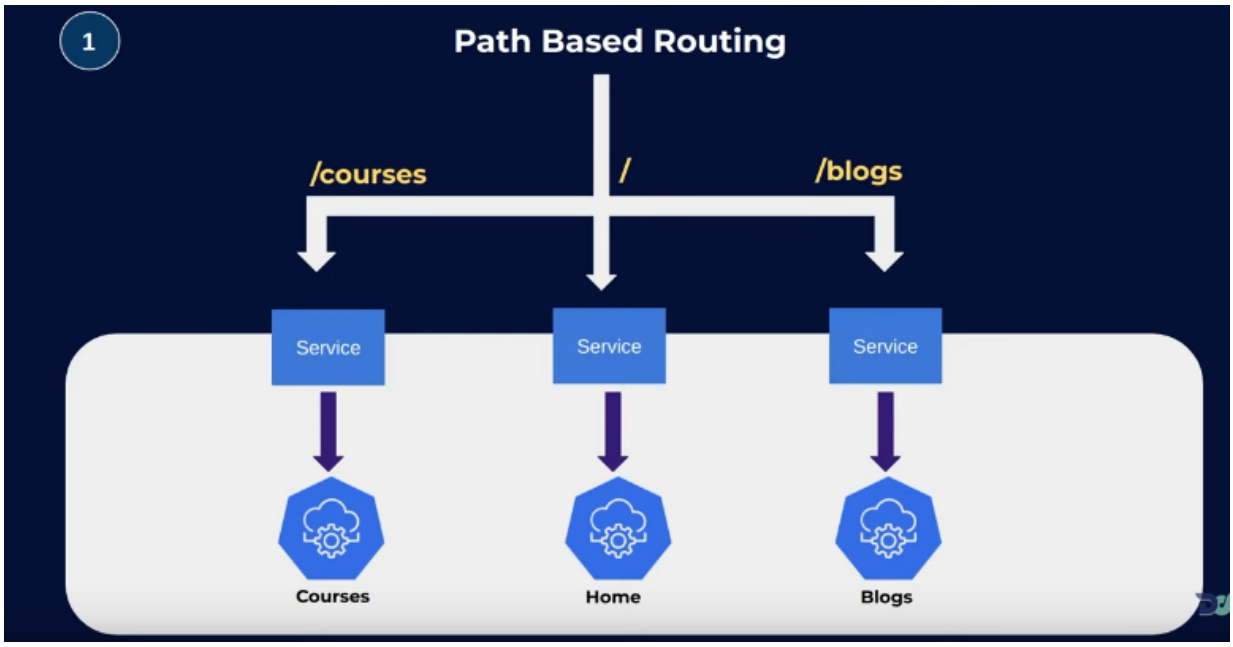
In this diagram:
- An Ingress Controller is responsible for fulfilling the Ingress rules and managing the external traffic.
- Services A and B represent the backend services or applications running in the Kubernetes cluster.

When a request comes from an external client, the Ingress Controller receives it and applies the defined rules to determine how to route the traffic. The Ingress Controller can perform various tasks, such as load balancing, SSL/TLS termination, and name-based virtual hosting.

Nginx ingress controller:

- Path-based routing is a method of routing traffic based on the path specified in the URL. In NGINX Ingress, you can define multiple Ingress resources, each with a different path, to route traffic to different backend services.
- For example, you could configure an Ingress resource to route all requests with the path `/api` to a backend service handling API requests, and requests with the path `/app` to a different backend service serving the application.
- This allows you to decouple your application's front-end from its back-end, making it easier to scale and manage your application.

Path Based Routing:

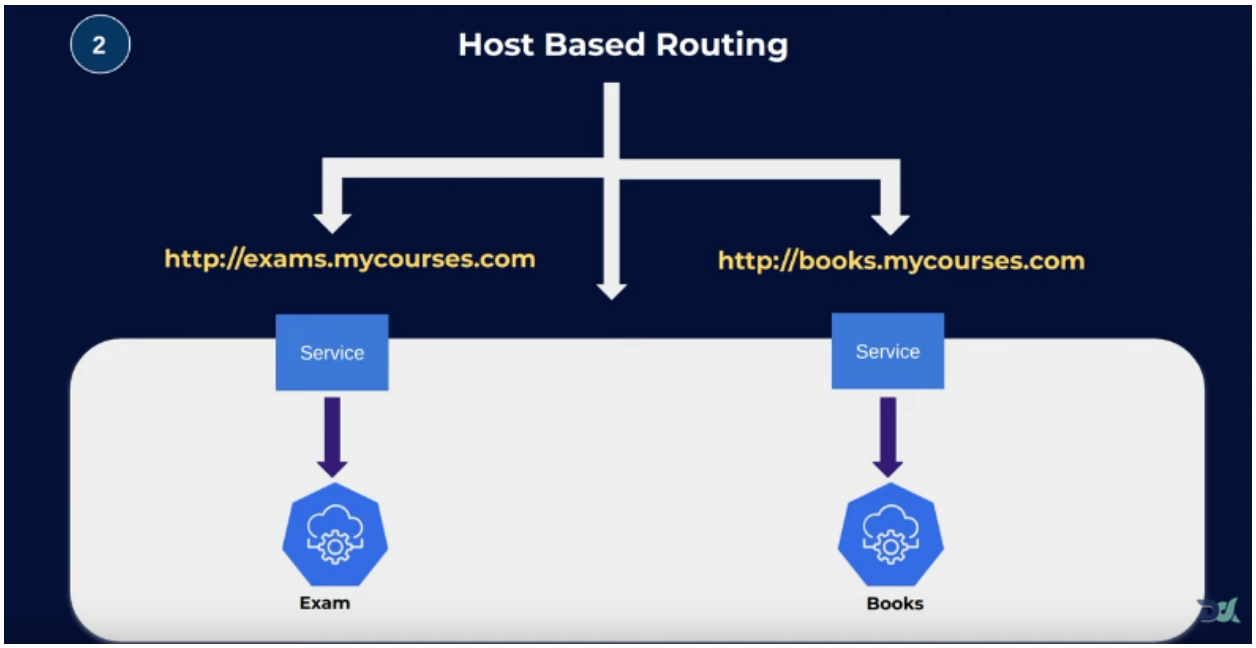


Here is an example of path-based routing in NGINX Ingress:

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: my-ingress
spec:
  rules:
  - host: mydomain.com
    http:
      paths:
      - path: /api
        backend:
          serviceName: api-service
          servicePort: 80
      - path: /app
        backend:
          serviceName: app-service
          servicePort: 80
```

This Ingress resource will route all requests to the `/api` path to the `api-service` service, and all requests to the `/app` path to the `app-service` service.

Host based Routing :



Host-based routing in nginx ingress refers to the routing of incoming traffic based on the hostname specified in the HTTP request. With host-based routing, you can configure different backend services or applications to handle requests for different hostnames.

When a request comes in, nginx ingress examines the "Host" header in the HTTP request to determine the hostname. It then uses this information to route the request to the appropriate backend service or application based on the defined rules.

For example, let's say you have two applications, App1 and App2, running on different backend services. You can configure nginx ingress to route requests for app1.example.com to the backend service serving App1, and requests for app2.example.com to the backend service serving App2. This allows you to host multiple applications on the same IP address or load balancer, using different hostnames to differentiate between them.

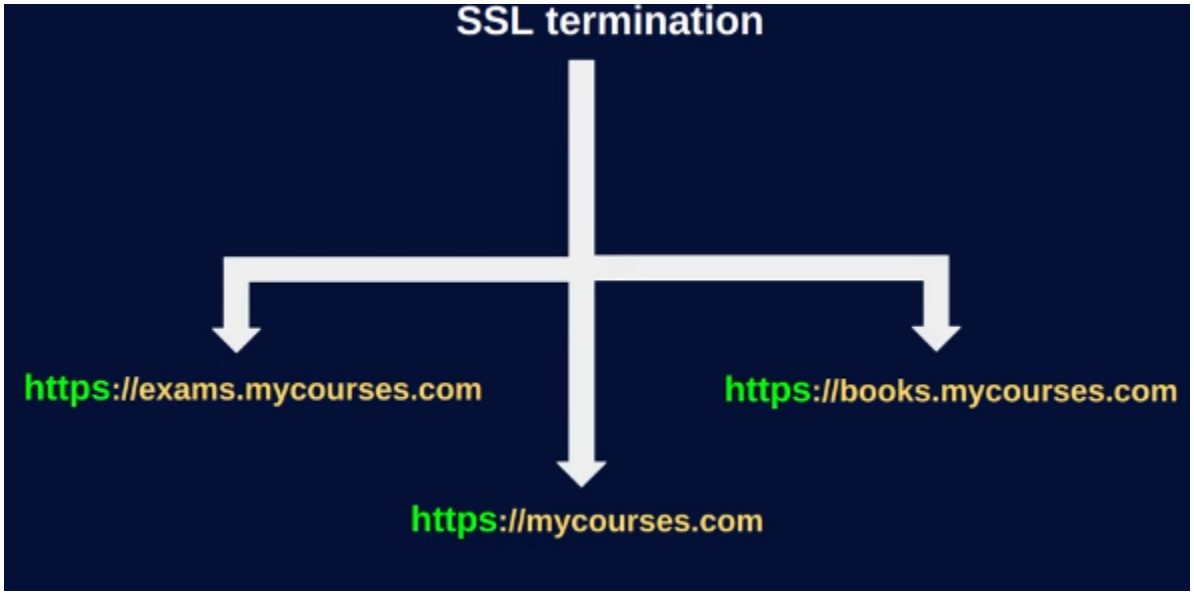
Host-based routing is useful in scenarios where you have multiple applications or services running on the same infrastructure and you want to route traffic based on the requested hostname.

To configure host-based routing in nginx ingress, you can use annotations in the Ingress resource definition. For example:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: my-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
  - host: app1.example.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: app1-service
            port:
              number: 80
  - host: app2.example.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: app2-service
            port:
              number: 80
```

In the above example, requests for app1.example.com will be routed to the "app1-service" backend service, and requests for app2.example.com will be routed to the "app2-service" backend service.

SSL Termination:



In nginx ingress, SSL termination can be configured to handle incoming SSL/TLS encrypted traffic and terminate the SSL connection at the ingress controller. This allows the ingress controller to decrypt the traffic and forward it to the backend services in plain HTTP.

To configure SSL termination in nginx ingress, you need to perform the following steps:

1.Obtain an SSL certificate: First, you need to obtain an SSL certificate for your domain or subdomain from a trusted certificate authority (CA). This can be done by generating a certificate signing request (CSR) and submitting it to the CA.

2.Create a Kubernetes Secret: Once you have the SSL certificate, you need to create a Kubernetes Secret to store the certificate and private key. This Secret will be referenced in the Ingress resource configuration.

3.Configure Ingress resource: Modify your Ingress resource definition to include the necessary annotations and TLS configuration. The annotations specify that SSL termination should be enabled and the Secret name to use for the SSL certificate.

Here's an example of an Ingress resource configuration for SSL termination in nginx ingress:

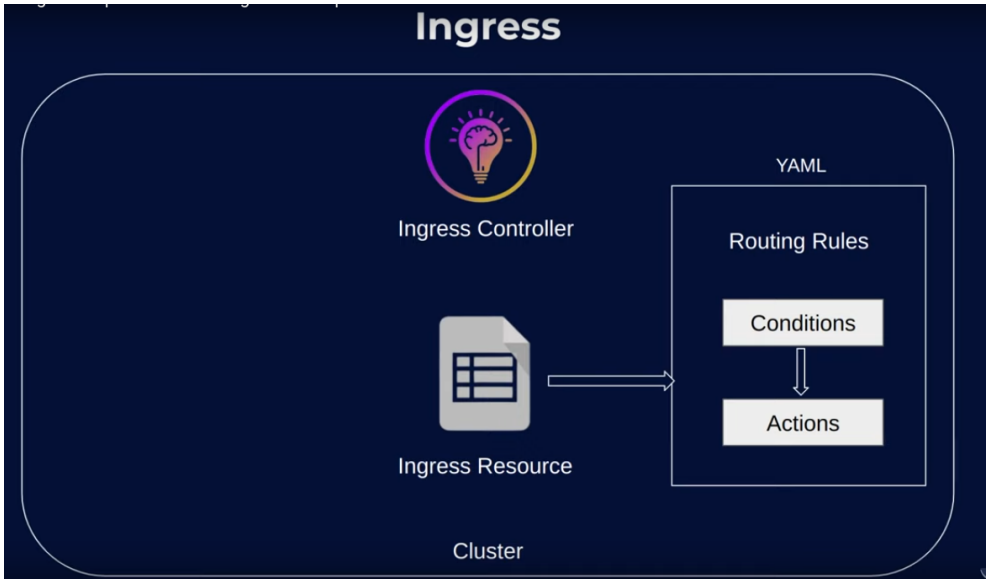
```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: my-ingress
  annotations:
    nginx.ingress.kubernetes.io/ssl-redirect: "true"
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  tls:
  - hosts:
    - example.com
    secretName: my-tls-secret
  rules:
  - host: example.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: my-backend-service
            port:
              number: 80
```

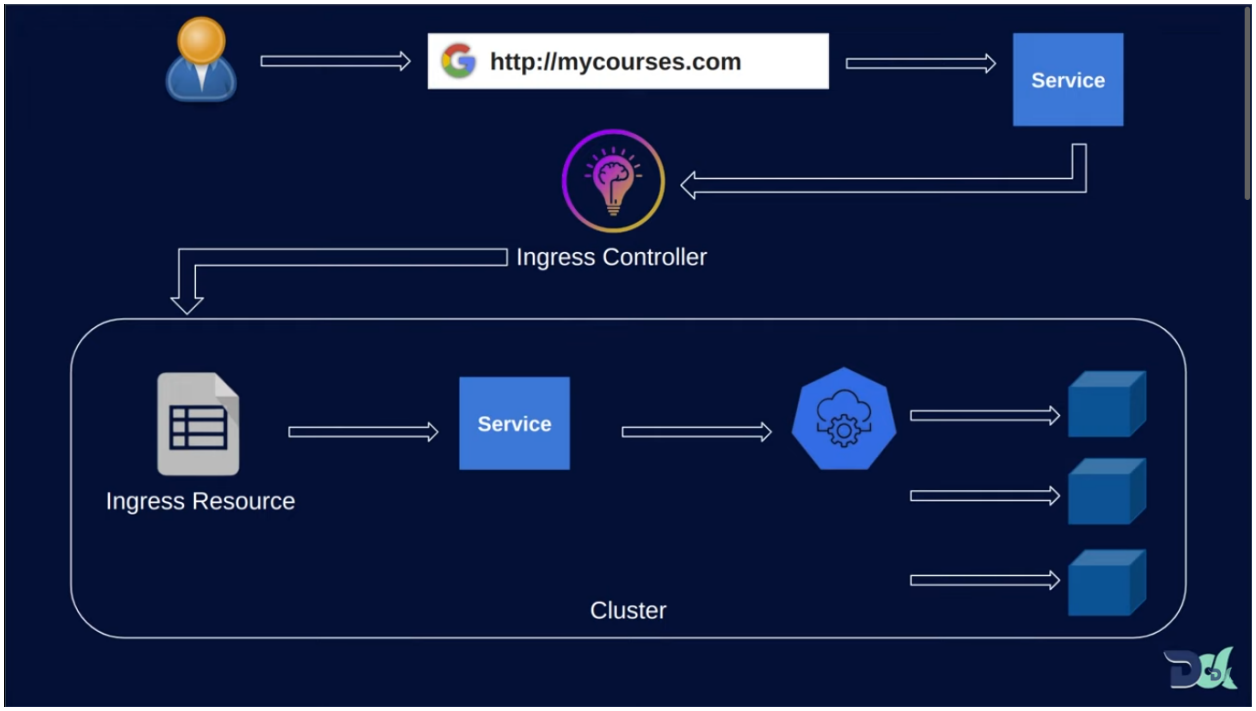
In the above example,

SSL termination is enabled with the `nginx.ingress.kubernetes.io/ssl-redirect: "true"` annotation. The `tls` section defines the hostnames for which SSL termination should be applied and the Secret (`my-tls-secret`) that contains the SSL certificate. The `rules` section specifies the host and backend service configuration.

Make sure to replace `example.com` with your actual domain or subdomain and `my-tls-secret` with the name of your Kubernetes Secret.

With this configuration, incoming SSL traffic for the specified hostname will be decrypted by the nginx ingress controller and forwarded to the backend service over plain HTTP.





Ingress Resource File

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: example-ingress
spec:
  rules:
  - host: "mycourses.com"
    http:
      paths:
      - pathType: Prefix
        path: /
        backend:
          service:
            name: home-service
            port:
              number: 8080
      - pathType: Prefix
        path: /courses
        backend:
          service:
            name: course-service
            port:
              number: 8181
```

steps to install the NGINX Ingress Controller on bare metal using manifest files, on bare-metal

Ref: https://kubernetes.github.io/ingress-nginx/deploy/#bare-metal-clusters

1. **Create a namespace:**

```
kubectl create namespace ingress-nginx
```

2. **Apply the mandatory resources:**

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/controller-v1.8.1/deploy/static/provider/baremetal/deploy.yaml
```

3. **Verify the installation:**

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

This command will display the running pods of the NGINX Ingress Controller. Wait until all the pods are in the "Running" state.

MetalLb :

MetalLB is a load-balancer implementation for bare metal Kubernetes clusters, using standard routing protocols.

Why ?

Kubernetes does not offer an implementation of network load balancers (Services of type LoadBalancer) for bare-metal clusters. The implementations of network load balancers that Kubernetes does ship with are all glue code that calls out to various IaaS platforms (GCP, AWS, Azure…). If you're not running on a supported IaaS platform (GCP, AWS, Azure…), LoadBalancers will remain in the "pending" state indefinitely when created.

Bare-metal cluster operators are left with two lesser tools to bring user traffic into their clusters, "NodePort" and "externalIPs" services. Both of these options have significant downsides for production use, which makes bare-metal clusters second-class citizens in the Kubernetes ecosystem.

MetalLB aims to redress this imbalance by offering a network load balancer implementation that integrates with standard network equipment, so that external services on bare-metal clusters also "just work" as much as possible.

Some key points to note about Metallb:

1. Purpose: Metallb is designed for bare metal Kubernetes clusters where the cloud provider's LoadBalancer service is not available. It enables you to allocate external IP addresses to services in your cluster and load balance traffic to those services.

2. IP Address Management: Metallb provides a range of IP addresses that can be used by the LoadBalancer services. You can configure this IP address range based on your network setup and requirements.

3. Layer 2 and BGP Modes: Metallb supports two different operation modes, Layer 2 and BGP. In Layer 2 mode, Metallb uses ARP announcements to claim IP addresses within the cluster's network. In BGP mode, it uses the Border Gateway Protocol to advertise the IP addresses to the network routers.

4. Configuration: Metallb is configured through a ConfigMap in Kubernetes. You can define the IP address range, protocol, and other settings in this ConfigMap. Changes to the ConfigMap are automatically applied to the load balancers.

5. Integration with Ingress Controller: Metallb can be used in conjunction with an Ingress Controller, such as NGINX Ingress Controller, to provide external access to services within the cluster. The Ingress Controller can utilize the LoadBalancer service created by Metallb to route traffic to the appropriate pods.

6. High Availability: Metallb supports multiple instances of the controller to provide high availability. You can run multiple controller instances in your cluster to ensure that load balancing functionality is not affected if one instance goes down.

7. Security Considerations: When exposing services through LoadBalancer services, it's essential to consider security implications. Ensure that appropriate firewall rules and network security measures are in place to protect the exposed services.

Metallb is a powerful tool for managing load balancing in bare metal Kubernetes clusters. It provides the functionality typically offered by cloud providers' LoadBalancer services, enabling you to expose services externally and distribute traffic efficiently.

How to deploy & use MetalLB in bare metal Kubernetes

- Checking kubectl version
    - kubectl version --short

Installation:

Ref link:

- https://metallb.universe.tf/

Check network interfaces

- ip a

```
vishal@vishal-VirtualBox:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 08:00:27:38:ec:a8 brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.7/24 brd 192.168.1.255 scope global dynamic noprefixroute enp0s3
       valid_lft 84246sec preferred_lft 84246sec
    inet6 fe80::440f:7d77:b1c:a72f/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default
    link/ether 02:42:29:ff:c3:46 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
       valid_lft forever preferred_lft forever
4: datapath: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1376 qdisc noqueue state UNKNOWN group default qlen 1000
    link/ether 2a:61:fa:ba:31:01 brd ff:ff:ff:ff:ff:ff
    inet6 fe80::2861:faff:feba:3101/64 scope link
```

- Check k8s nodes range
    - kubectl get nodes -o wide

```
vishal@vishal-VirtualBox:~$
vishal@vishal-VirtualBox:~$ kubectl get nodes -o wide
NAME               STATUS   ROLES           AGE   VERSION   INTERNAL-IP    EXTERNAL-IP   OS-
TIME
manoj              Ready    <none>          35m   v1.27.3   192.168.1.11   <none>        Ubu
1.6.12
vishal-virtualbox  Ready    control-plane   36m   v1.27.3   192.168.1.7    <none>        Ubu
1.6.12
vishal@vishal-VirtualBox:~$
```

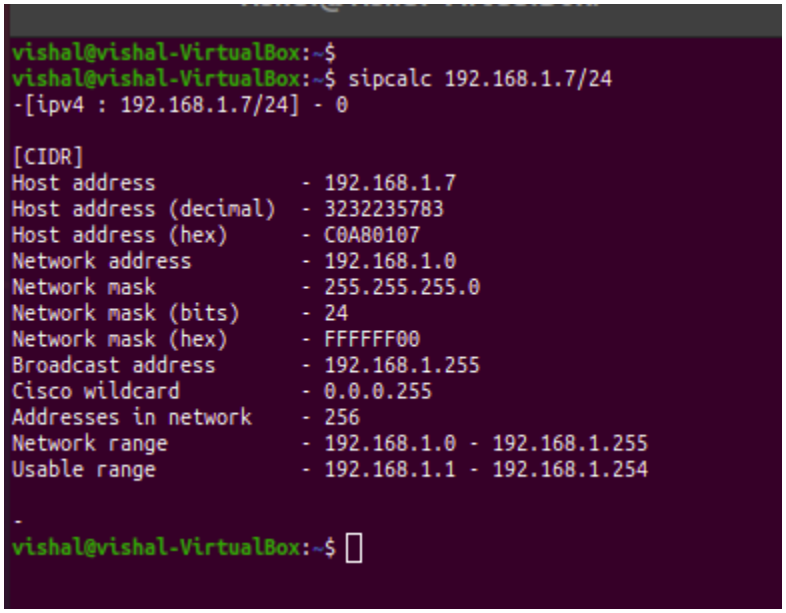
Setup ip address in this network:

- Install sipcalc if you dont have it
  - sudo apt-get install sipcalc -y

Analyze the network ip address range

In my case it is: 192.168.1.7/24

- sipcalc 192.168.1.7/24



Network range - 192.168.1.0 - 192.168.1.255
Usable range - 192.168.1.1 - 192.168.1.254

As you can see my usable range. So,

I m going to assign 20 ip address to the metallb

I m picking up : 192.168.1.30 - 192.168.1.50

Install metal lb using manifests:

- kubectl apply -f https://raw.githubusercontent.com/metallb/metallb/v0.13.10/config/manifests/metallb-native.yaml

Configure ip address range for metallb

Ref : https://metallb.universe.tf/configuration/

Layer 2 configuration

Copy and edit the file

- mkdir metallb
- cd metallb
- vi ipaddresspool.yml
- Specify ip address range
- Save,exit and apply

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  name: first-pool
  namespace: metallb-system
spec:
  addresses:
  - 192.168.1.30-192.168.1.50
```

Edit next

- vi l2-adv.yml
- Save,exit,apply

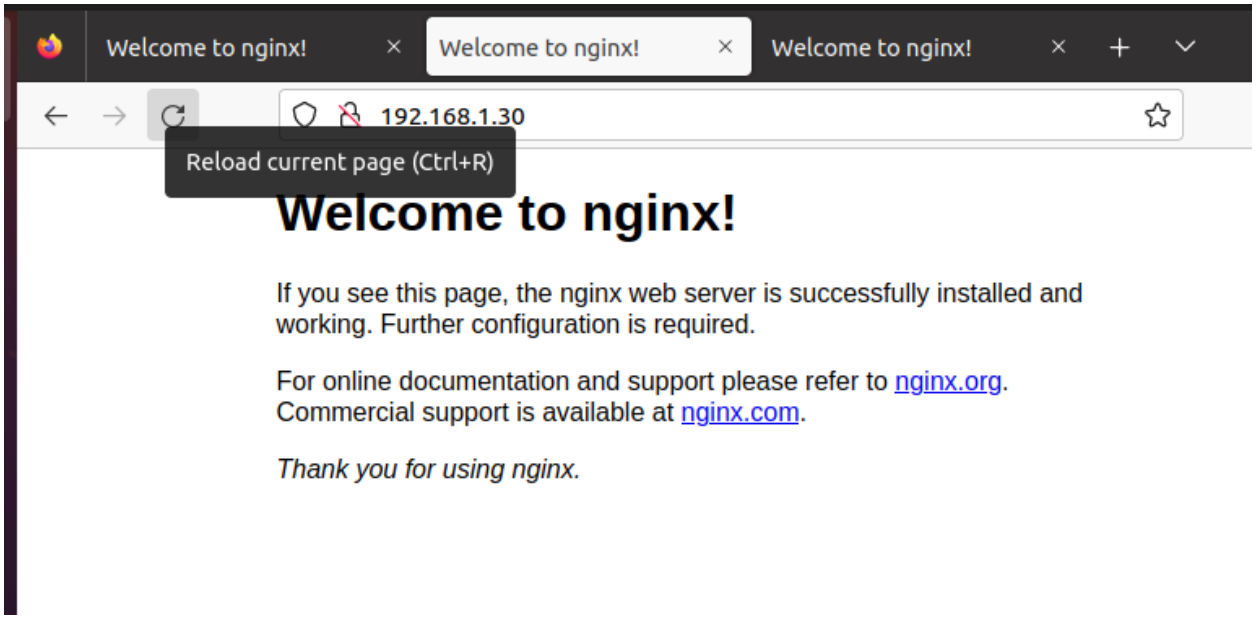
```
apiVersion: metallb.io/v1beta1
kind: L2Advertisement
metadata:
  name: l2-adv
  namespace: metallb-system
spec:
  ipAddressPools:
  - first-pool
```

Now we will have external ip address for our application:

```
vishal@vishal-VirtualBox:~/metallb$
vishal@vishal-VirtualBox:~/metallb$ kubectl get all
NAME                                 READY   STATUS    RESTARTS   AGE
pod/nginx-deployment-f79c9cccd-p46bd   1/1     Running   0          77m

NAME                    TYPE           CLUSTER-IP       EXTERNAL-IP    PORT(S)        AGE
service/kubernetes      ClusterIP      10.96.0.1        <none>         443/TCP        84m
service/nginx-service   LoadBalancer   10.100.193.175   192.168.1.30   80:30463/TCP   77m

NAME                               READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/nginx-deployment   1/1     1            1           77m

NAME                                          DESIRED   CURRENT   READY   AGE
replicaset.apps/nginx-deployment-f79c9cccd    1         1         1       77m
vishal@vishal-VirtualBox:~/metallb$
```

Check it using ip we have got,

Paste ip in the browser,



Other References:

Nginx ingress: (path and hostpath base routing and tls termination)

- https://www.youtube.com/watch?v=-2VKSYffdYM&t=10s
- https://www.youtube.com/watch?v=pcADx8JFUIA&t=95s

Metal Lb:

- https://www.youtube.com/watch?v=2SmYjj-GFnE

My things:

Sourcecode :

- https://github.com/vishalk17/devops/tree/main/kubernetes

My devops repo :

- https://github.com/vishalk17/devops

My telegram channel:

- https://t.me/vishalk17_devops

Contact:

Telegram : t.me/vishalk17

vishalk17 My youtube Channel :

- https://www.youtube.com/@vishalk17