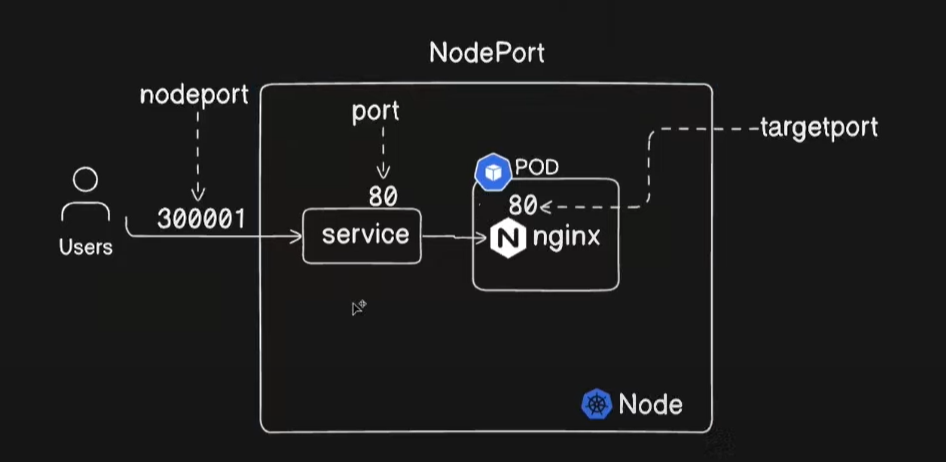
**Each pod associate with its own IP but these IP are unstable in nature, whenever a Pod restarts or autoScales, the new IP will be replacing with old one, making Pod-Pod communication much worse.**

**We need a kind which actually provides a persistent IP address throughout the life cyle of the PODs.**

**Service:** A service in kubernetes is a kind which actually does two major functionalities.

1. Establish stable network communication between the pods and serves the traffic from internet.
2. It comes up with auto load balancing properties which routes the traffic evenly to the all target pods.

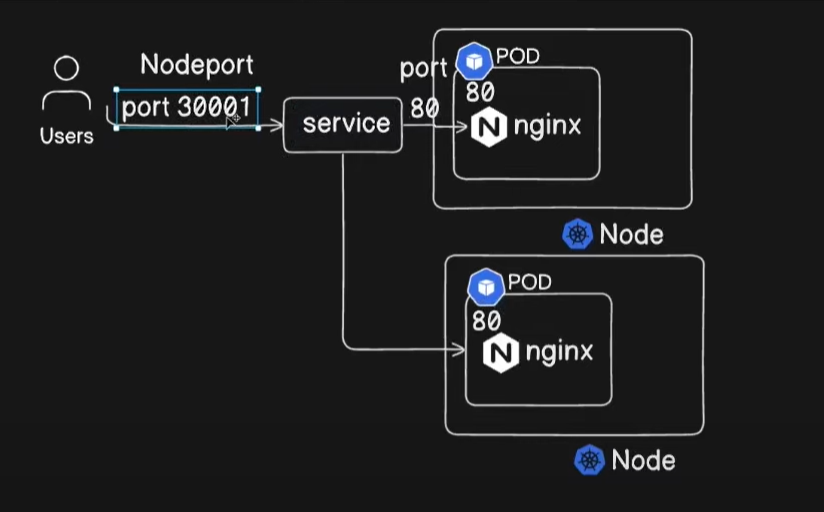


**NodePort:** it is a port on node which we expose to outside network for frontend application to serve the internet traffic. Ex: <http://12.344.567.54:30001>

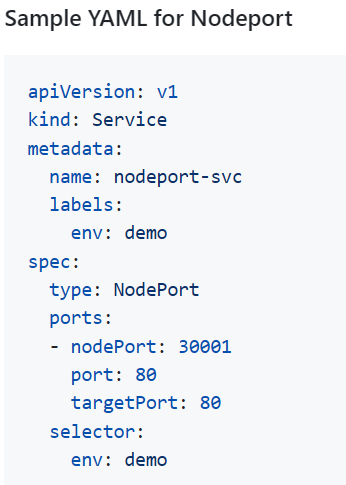
Node port range should be in **30,000 – 32,767**.

**Service Port**: for each service we would be assigning a port for internal communication not with outside internet. For example if other pods want to connect with your frontend application pod internally we use this port.

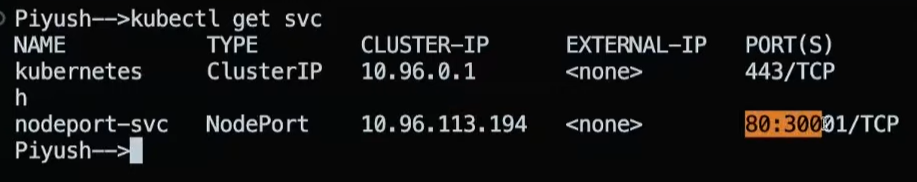
**Target Port:** whether the traffic is coming from **Nodeport/ServicePort**, it will directed/forwared to the target port where our Pod is actually running.



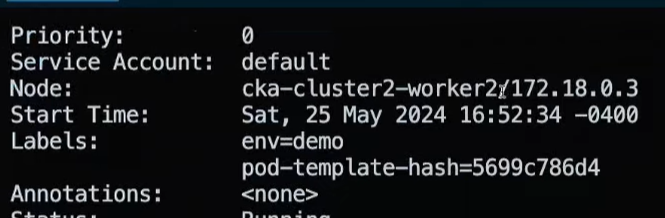
A same service can be assigned to multiple pods, the traffic is distributed to multiple pods on round robin method with auto load balancing.



* ***Kubectl create –f nodeport.yaml***

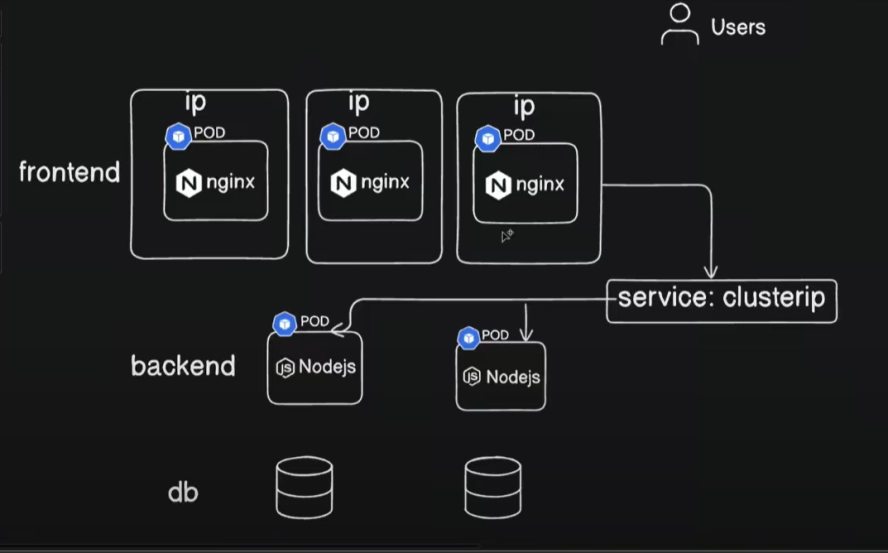
******

* ***Kubectl describe pod <podName>*** 🡪 to get the node in which you are pod is running



* ***Curl*** [***http://172.18.0.3:30001***](http://172.18.0.3:30001) ***🡪 take the nodeIP and hit the URL, the same can be accessible via browser as well.***

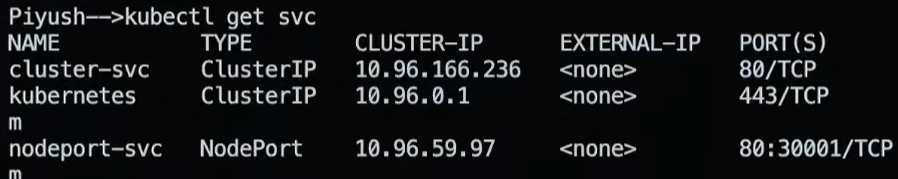
******

******

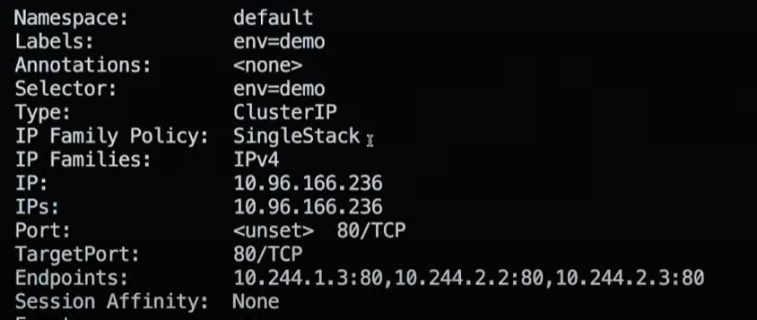
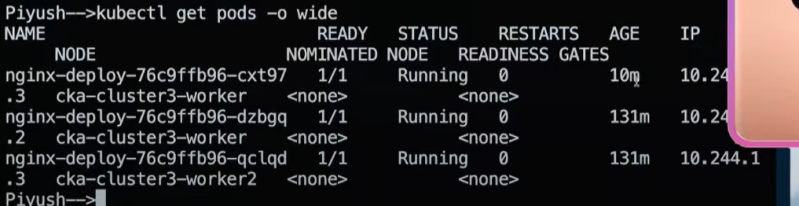
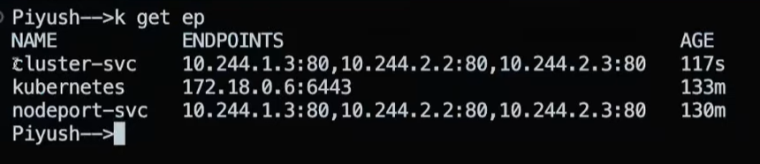
***ClusterIP Service: It establishes internal network communication between the pods as pods IP address are unstable in nature, the pods which are assigned with service shares the new IP addresses automatically with cluserIP service to maintain stablilty.***

***Here we assign Pods to service with pod labels only.***

* ***Kubectl create –f CluserIP.yaml***

******

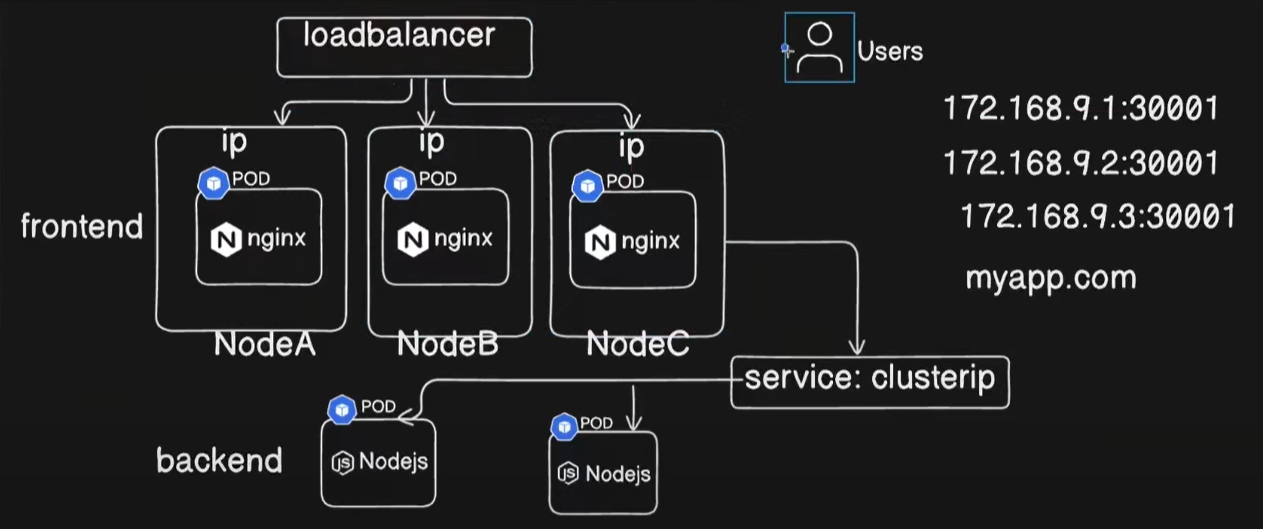
* ***Kubectl describe svc <clusterIP\_SVC>***

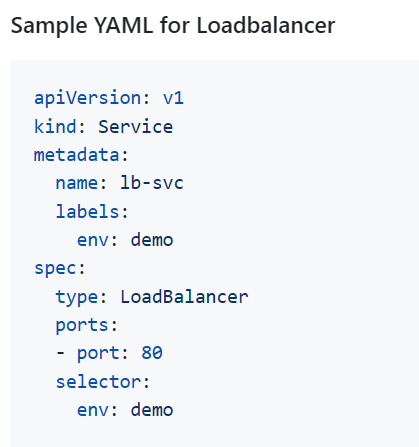
*** ***

***IP:*** *this is the actual stable IP address which is used for pods mutual communication*

***EndPoints:*** *are the unstable IP addresses of the pods which are assigned with this service and these endpoints automatically updated with new once pods are restarted.*

***This is how, A clusterIP provides you a stable IP address by collecting the PODs IP info dynamically and maps to the end points.***

******

******

### External Service Load balancing

For external access, we require either a NodePort or LoadBalancer type of service.

We can use the NodePort service type if we have a limited number of services. It gives connectivity to our application without actually having a dedicated external load balancer.

Please keep in mind that it will work as long as the node is reachable via its IP addresses. In most cases, the worker nodes reside in our private network(like office networks or private VPC). In such cases, we can not access the NodePort service from the Internet.

Another disadvantage of the NodePort service type is that it creates a mapping to Node’s IP address on a static port. The allocatable port range is 30000–32767, and the service must allocate the same port on each node while provisioning. It becomes problematic when the application scales up into multiple microservices.

#### External Load balancing using LoadBalancer

The public cloud providers like AWS, GCP, Azure, etc., automatically create load balancers when creating a service with spec.type: LoadBalancer.

LoadBalancer type provides a Public IP address or DNS name to which the external users can connect. The traffic flows from the LoadBalancer to a mapped service on a designated port, which eventually forwards it to the healthy pods. Note that LoadBalancers doesn’t have a direct mapping to the pods.

aws ec2 create-key-pair --region us-west-2 --key-name myKeyPair

eksctl create cluster

--name myEKSCluster

--region ap-south-1

--with-oidc

--ssh-access

--ssh-public-key <myKeyPair-public-key-path>

--managed

Once we have our Kubernetes Cluster ready, we need to launch an RMQ pod. We will use the below Pod manifest, rmq-pod.yaml:

apiVersion: v1

kind: Pod

metadata:

name: rabbitmq

labels:

app: rabbitmq

spec:

containers:

- name: rabbitmq

image: rabbitmq:latest

Then create the pod:

kubectl apply -f rmq-pod.yaml

Verify the pod is up and running:

❯ kubectl get pods --selector='app=rabbitmq'

NAME READY STATUS RESTARTS AGE

rabbitmq 1/1 Running 0 91m

Now, let us create the Service manifest, rmq-svc.yaml:

apiVersion: v1

kind: Service

metadata:

name: rmq-external-service

spec:

type: LoadBalancer

selector:

app: rabbitmq

ports:

- name: rmq-admin

protocol: TCP

port: 15672

targetPort: 15672

Create the service:

❯ kubectl apply -f rmq-svc.yaml

service/rmq-external-service created

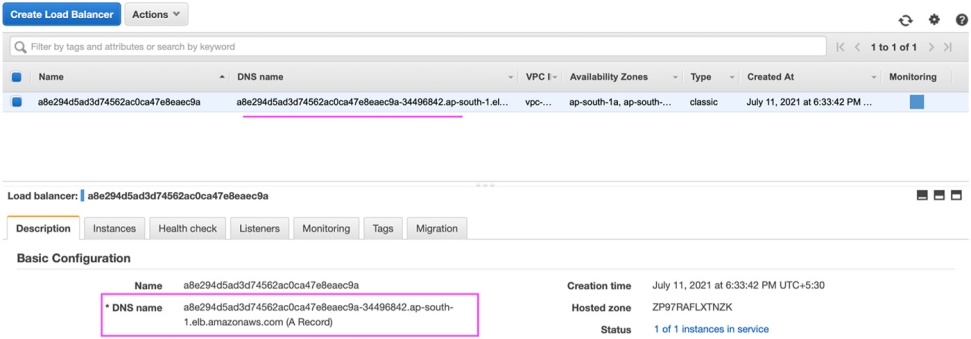
Verify the service:

❯ kubectl get svc

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

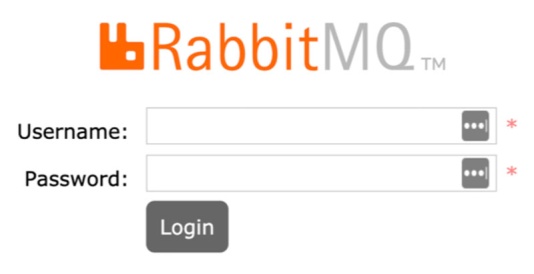
rmq-external-service LoadBalancer 10.100.37.235 a8e294d5ad3d74562ac0ca47e8eaec9a-34496842.ap-south-1.elb.amazonaws.com 15672:31716/TCP 4m45s

If you have noticed, we have an LB DNS name under EXTERNAL-IP. It is the ELB created in AWS as shown below:

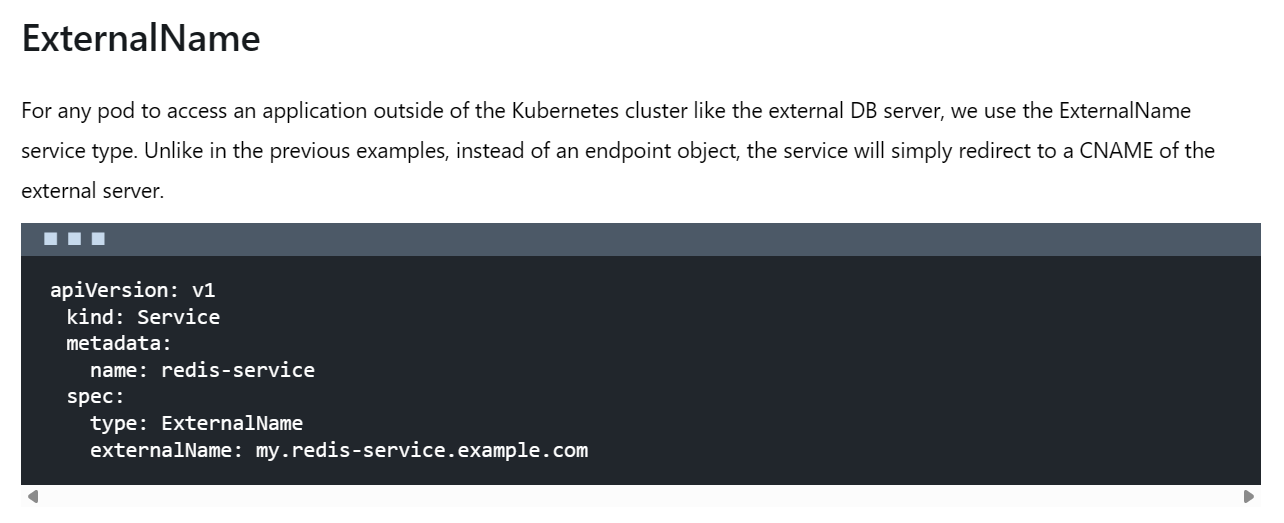


ELB under AWS

Copy the DNS name in a browser with port 15672. In our case, it would be<http://a8e294d5ad3d74562ac0ca47e8eaec9a-34496842.ap-south-1.elb.amazonaws.com:15672>



You can share this URL/DNS name with anyone who wants to have access to your RMQ Admin. We have seen how easy it is to create and configure an external Load Balancer to expose our application. However, there are some limitations in Load Balancer, which we will see in the next section, and how Ingress can help.

******

[***https://spacelift.io/blog/kubernetes-ingress***](https://spacelift.io/blog/kubernetes-ingress)