**Ingress Controller**

In Kubernetes, an Ingress Controller is a component that manages and operates the Ingress resource.

It is responsible for fulfilling the Ingress rules by configuring and managing the underlying load balancer or reverse proxy that handles the incoming traffic.

By using an Ingress Controller, Kubernetes clusters can leverage the Ingress resource to manage external access to services in a more centralized and flexible manner.

The Ingress Controller abstracts the complexities of load balancing and routing, providing a convenient interface to define and manage the ingress rules for applications running in the cluster.

**Ingress Resources**

In Kubernetes, an Ingress resource is an API object that defines rules for routing external HTTP and HTTPS traffic to services within the cluster.

It acts as a layer 7 (application layer) load balancer and allows for more advanced traffic routing and configuration compared to the basic Service and NodePort approaches.

Using Ingress resources, you can define complex routing rules, manage multiple backend services, and customize the behavior of incoming traffic for your applications in Kubernetes.

The Ingress Controller will handle the implementation and enforcement of these rules, allowing external traffic to be efficiently and securely routed to the appropriate services within the cluster.

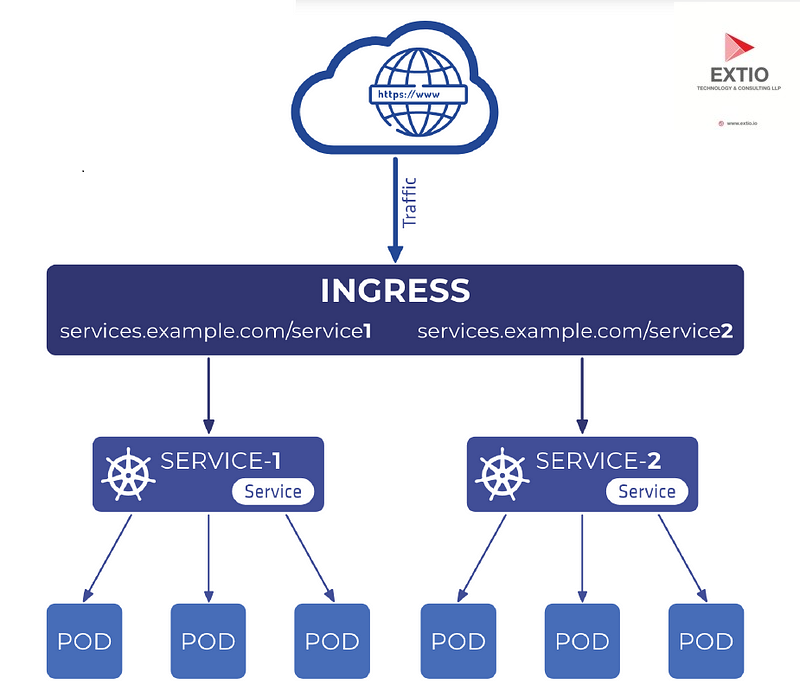
In Kubernetes, **Ingress** and **LoadBalancer Service** both handle exposing applications to external traffic, but they serve different purposes and use cases. Here's why you might choose to use **Ingress** over a **LoadBalancer Service**:

**1. Advanced Traffic Management**

* **Ingress** provides **layer 7 (application layer) routing**, enabling features like:
  + **Path-based routing**: Route traffic to different services based on URL paths (e.g., /app1 vs. /app2).
  + **Host-based routing**: Route traffic to different services based on hostnames (e.g., app1.example.com vs. app2.example.com).
  + SSL/TLS termination: Ingress can manage certificates to secure your traffic using HTTPS.
* In contrast, a **LoadBalancer Service** operates at **layer 4 (transport layer)**, forwarding traffic directly to a service's pods based on IP and port without inspecting or modifying the request.

**2. Cost-Effectiveness**

* **LoadBalancer Services** provision a cloud provider's load balancer for every service, which can become costly if you have multiple services.
* With **Ingress**, you can use a single Ingress controller (and one cloud load balancer) to manage routing for multiple services, reducing infrastructure costs.



**Ingress Demo**

**Install microk8s and setup**

sudo apt update -y

sudo snap install microk8s --classic --channel=1.31

sudo usermod -a -G microk8s $USER

mkdir -p ~/.kube  
chmod 0700 ~/.kube

newgrp microk8s

microk8s status --wait-ready

sudo snap install kubectl --classic

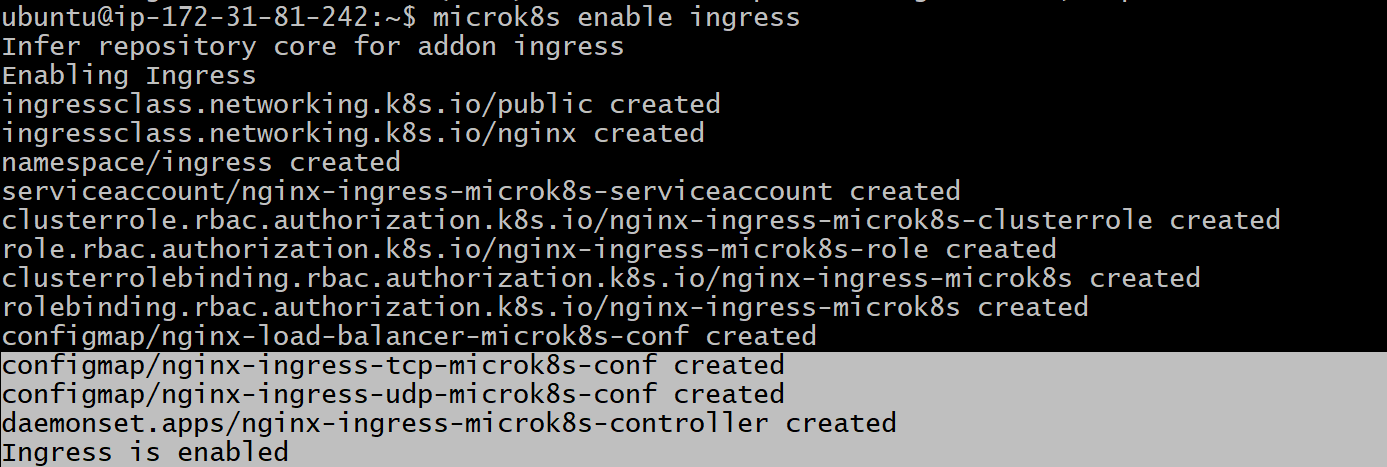
microk8s kubectl get services

alias kubectl='microk8s kubectl'

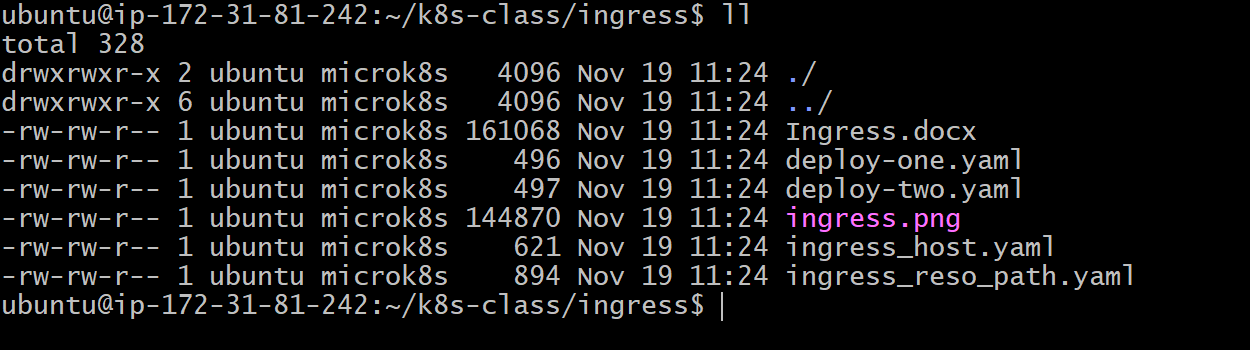
**git clone repo**

**url** [**https://github.com/vickydevo/k8s-class.git**](https://github.com/vickydevo/k8s-class.git)

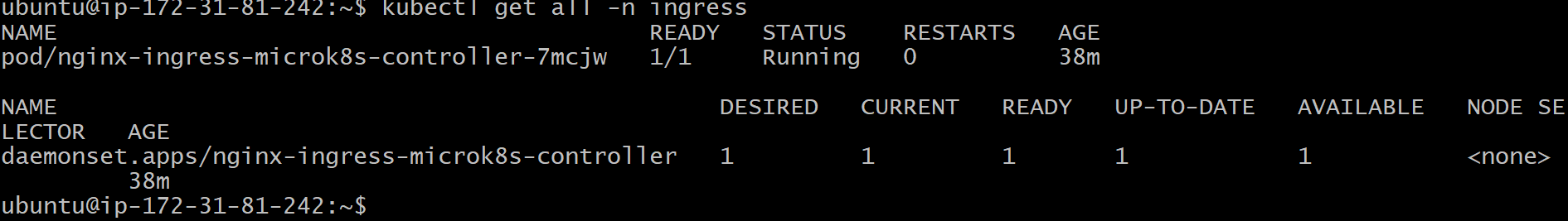
**enable ingress controller**

****

Check yaml files available or not

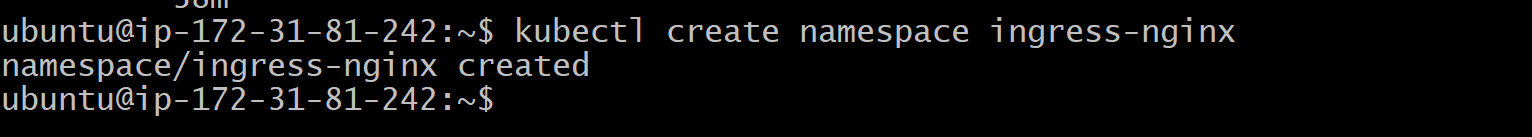


Call resources under namespace of ingress



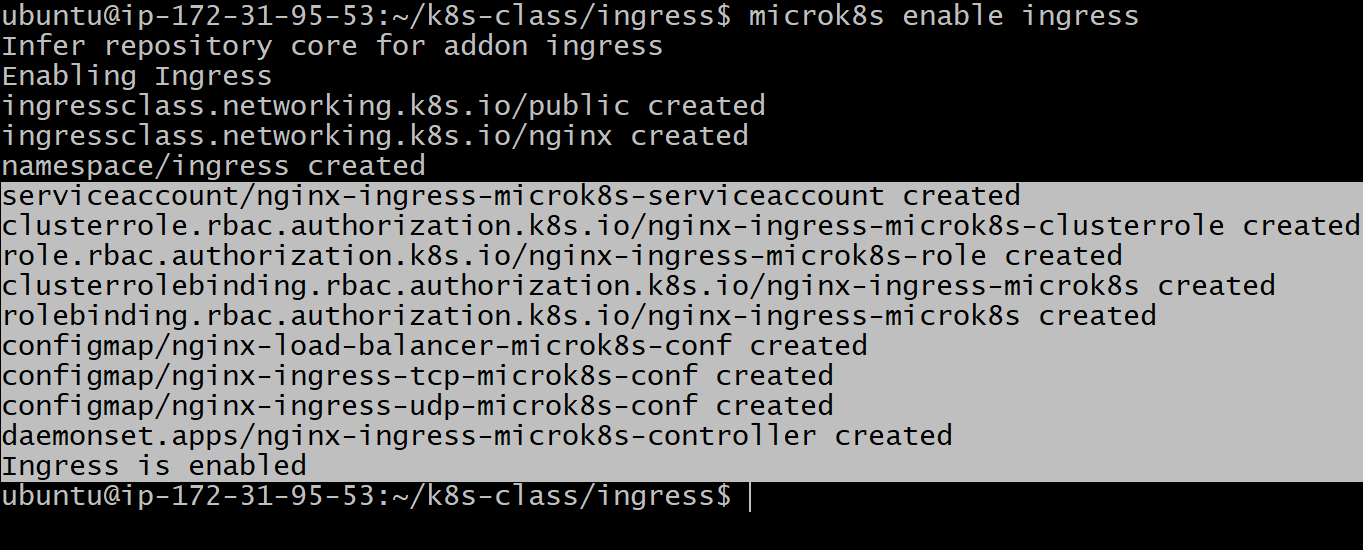
Create 3rd party ingress controller from nginx

First create namespace

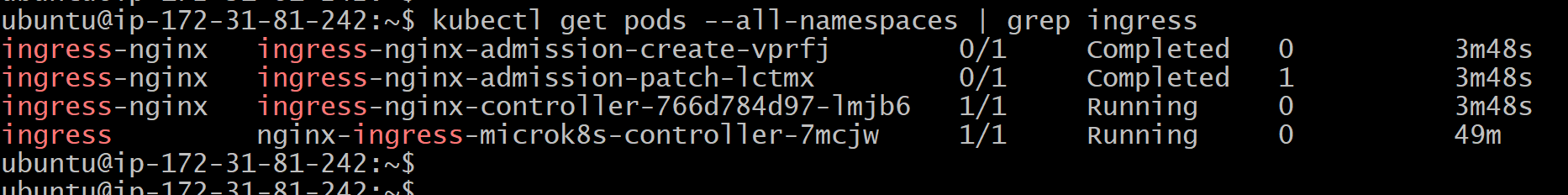


kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/controller-v1.2.1/deploy/static/provider/cloud/deploy.yaml

Enable microk8s ingress

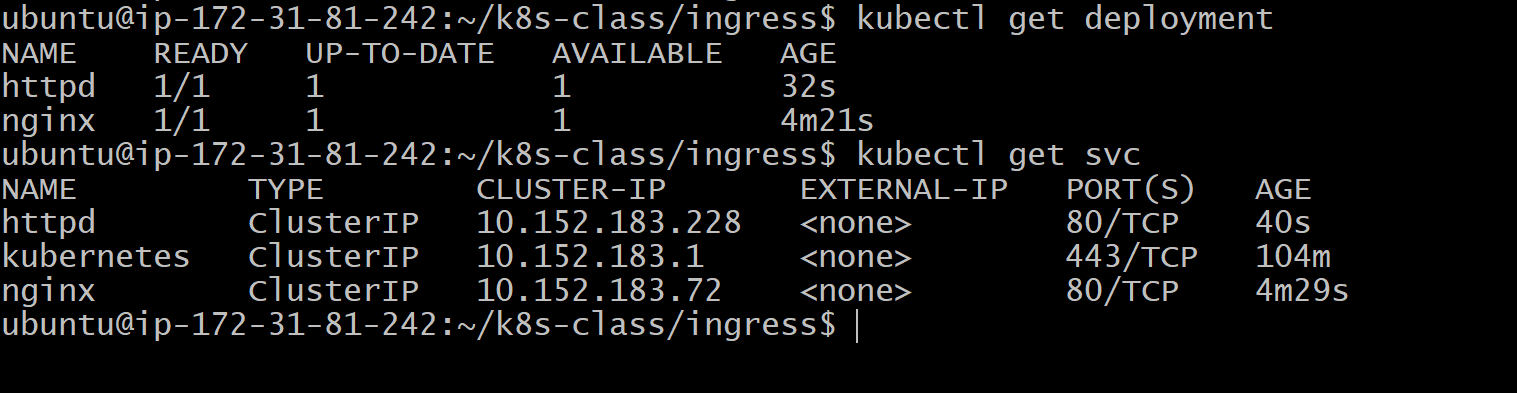


List Pods Across All Namespaces:



kubectl apply -f deploy-two.yaml

kubectl apply -f deploy-one.yaml



Change the host flag in ingress-host.yaml file replace with your **public ip**

