Ingress Overview

Services Recap:

- In Kubernetes, a Service is a method for exposing a network application that is running as one or more Pods in your cluster.

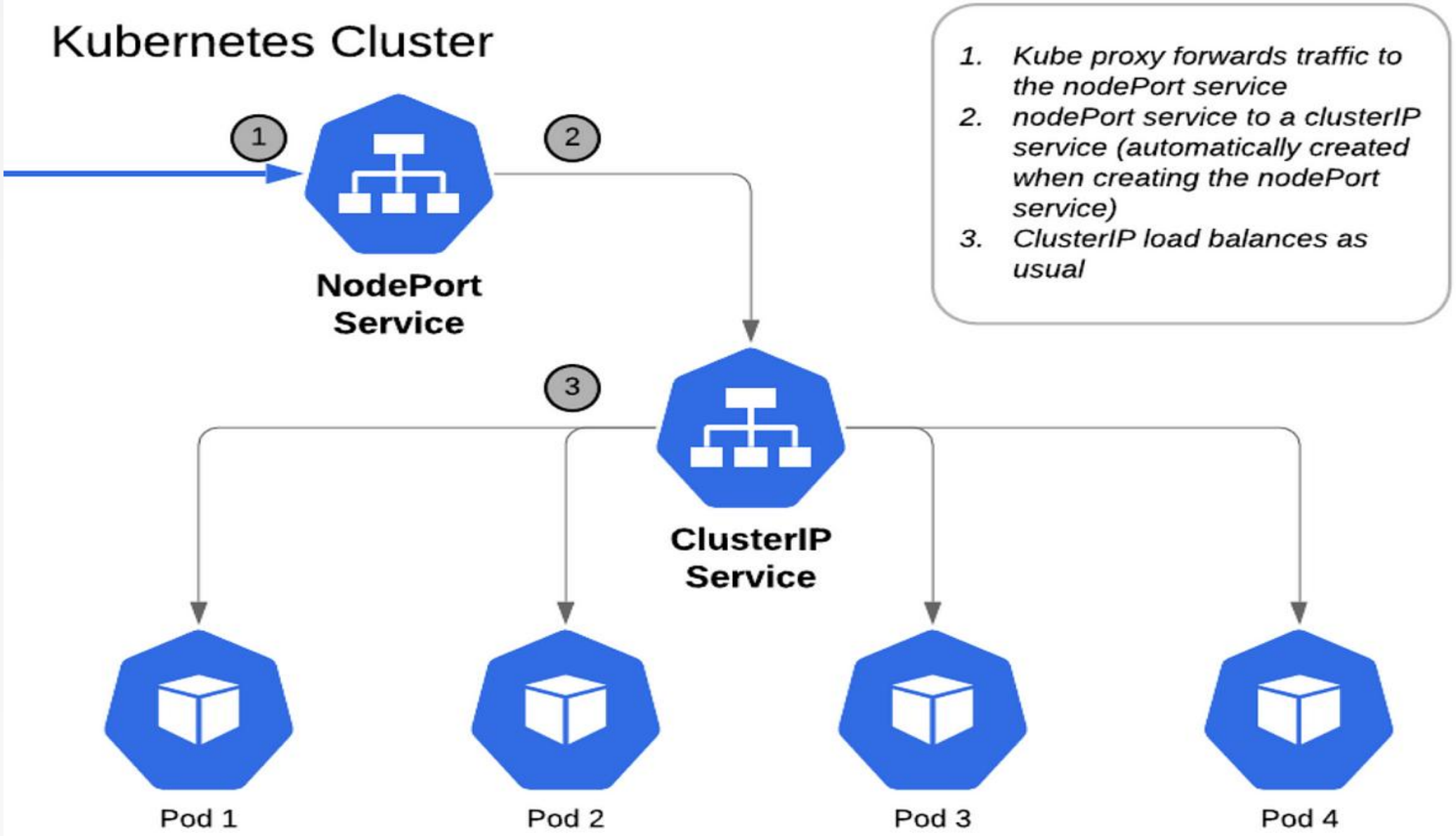
- It is of three main types:
    I. ClusterIP
    II. NodePort
    III. LoadBalancer

Type 1: ClusterIP

- Default service type in Kubernetes.

- ClusterIP Services are meant for Pod-to-Pod communication only.

  They aren't accessible from outside the cluster.


Kubernetes Cluster
1. Kube proxy forwards traffic to the nodePort service
2. nodePort service to a clusterIP service (automatically created when creating the nodePort service)
3. ClusterIP load balances as usual
1
2
NodePort Service
3
ClusterIP Service
Pod 1
Pod 2
Pod 3
Pod 4

Type 2: NodePort

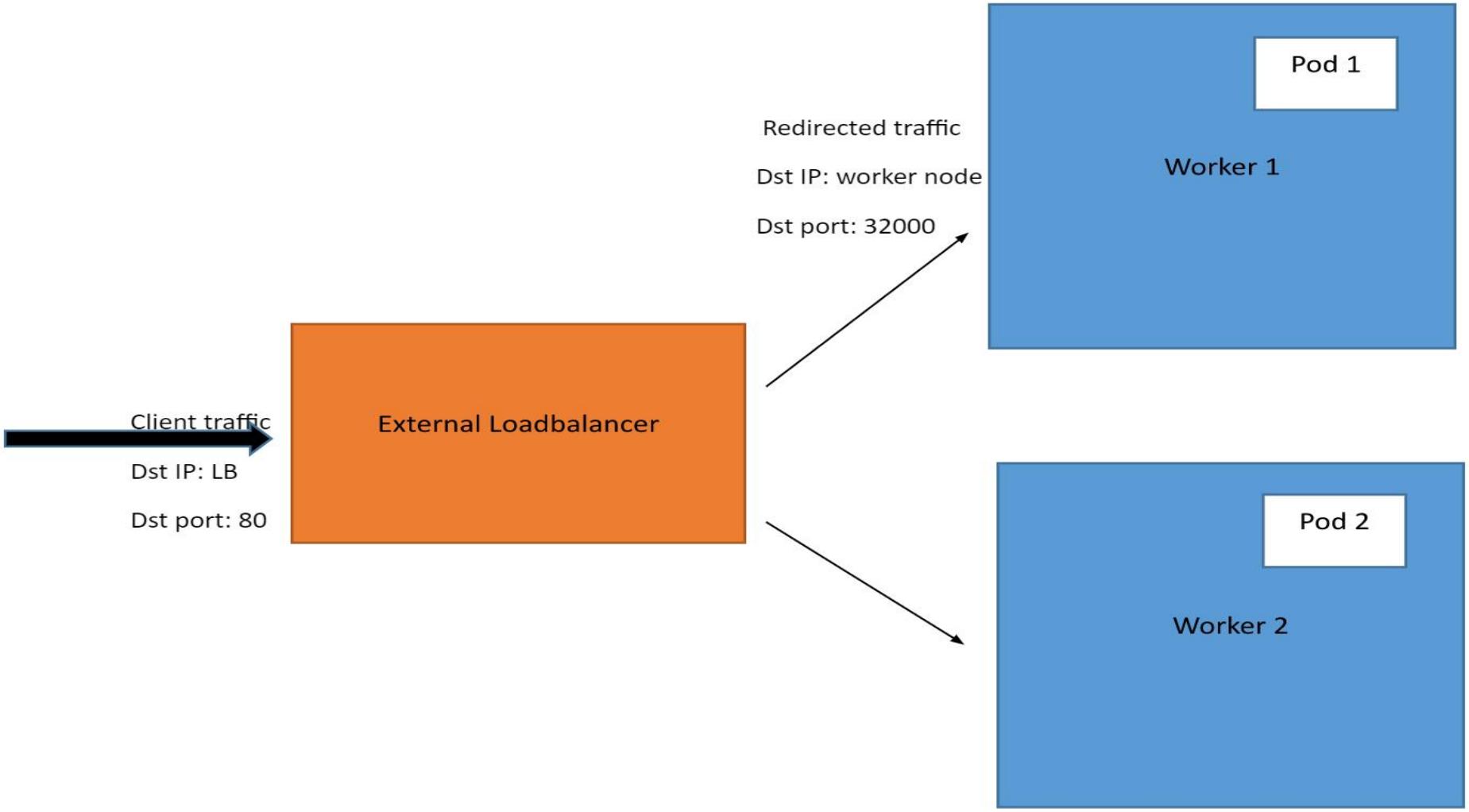
- It is useful when you need to expose your application to external clients. It builds on top of the ClusterIP service.
- When you create a NodePort Service, Kubernetes opens a port (in the range of 30000 and 32767) on all of its worker nodes.
- All traffic incoming to the worker node's IP address, and that specific port, is redirected to a Pod linked with that Service.

Type 3: LoadBalancer

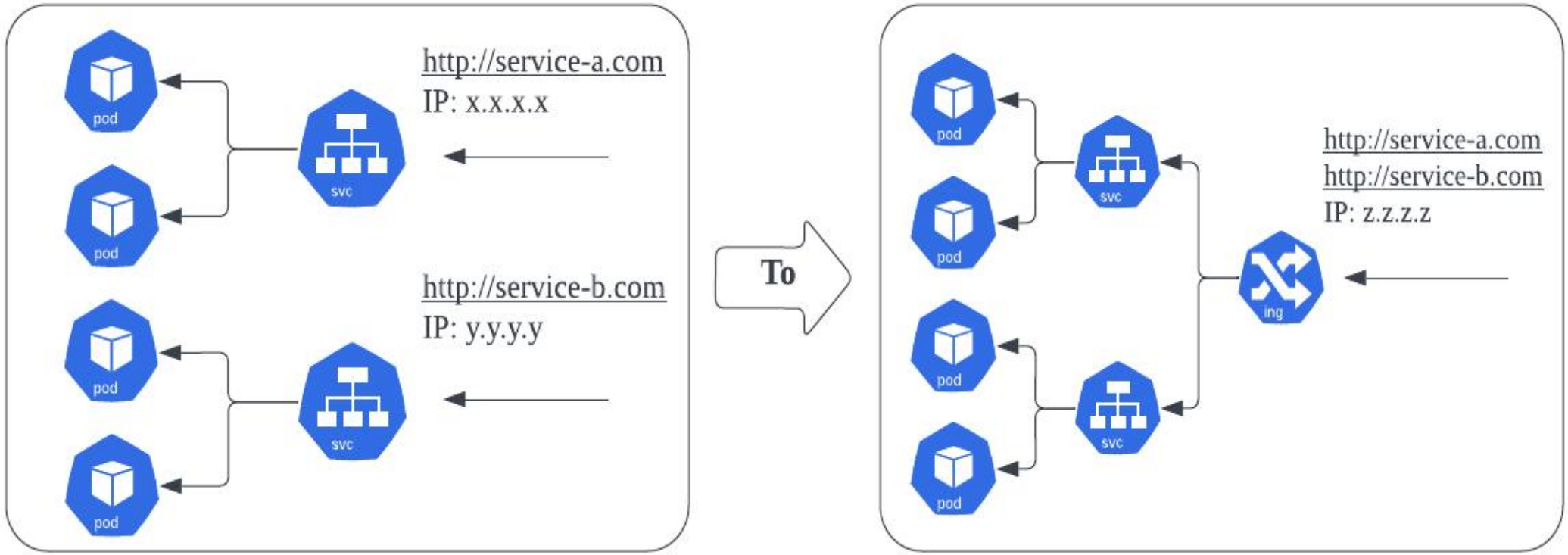
- It is based on the NodePort service and distributes network traffic across multiple instances of an application running in a K8S cluster.

- It adds the ability to configure external load balancers in public and private clouds. It exposes services running within the cluster by forwarding network traffic to cluster nodes.

- Load balancers in K8S can be implemented by using a cloud provider-specific load balancer such as Azure Load Balancer, AWS Network Load Balancer (NLB), or Elastic Load Balancer (ELB) that operates at the Network Layer 4 of the OSI model.


Pod 1
Redirected traffic
Dst IP: worker node
Dst port: 32000
Worker 1
Client traffic
Dst IP: LB
Dst port: 80
External Loadbalancer
Pod 2
Worker 2

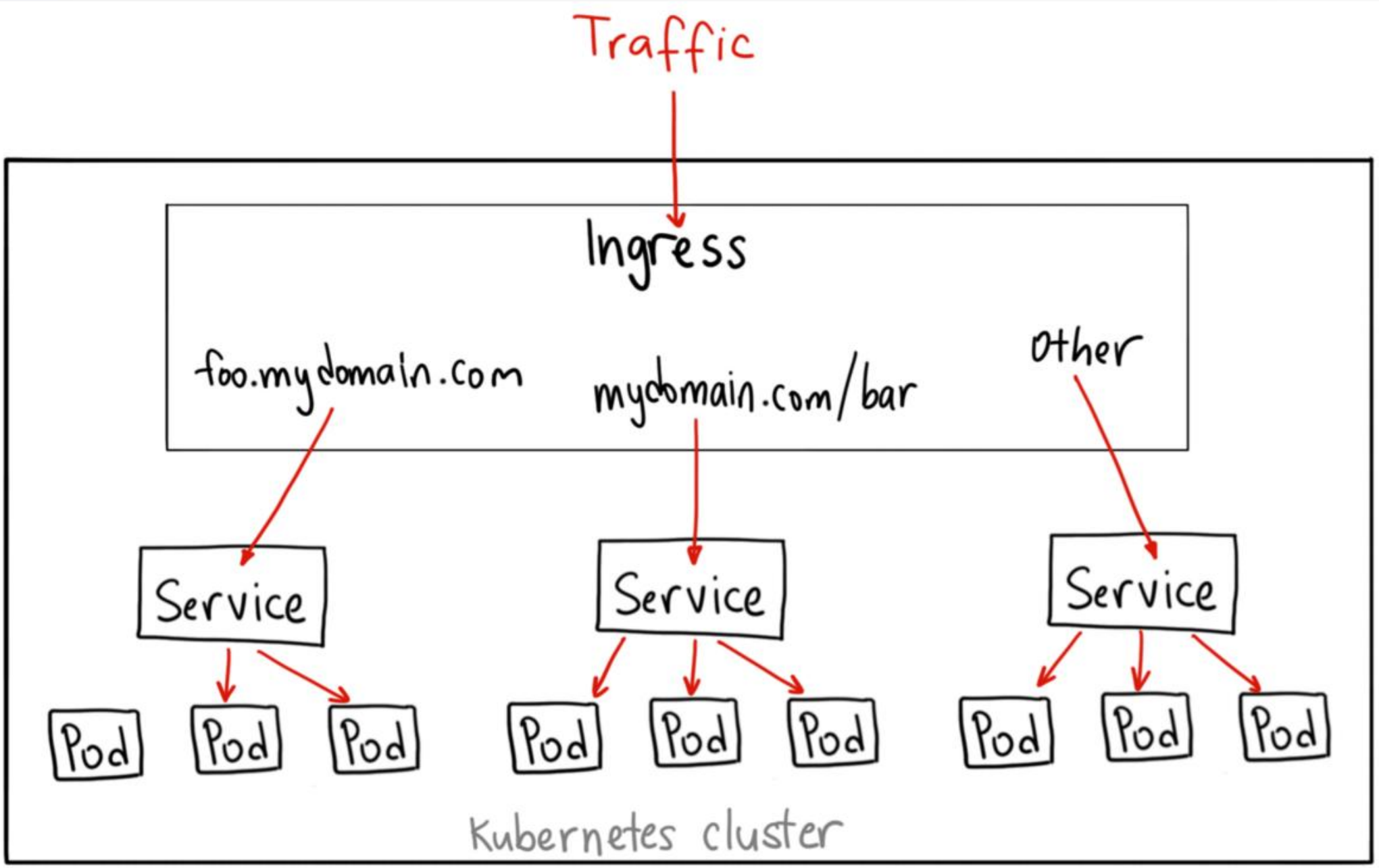
What if there are multiple applications running?



Ingress:

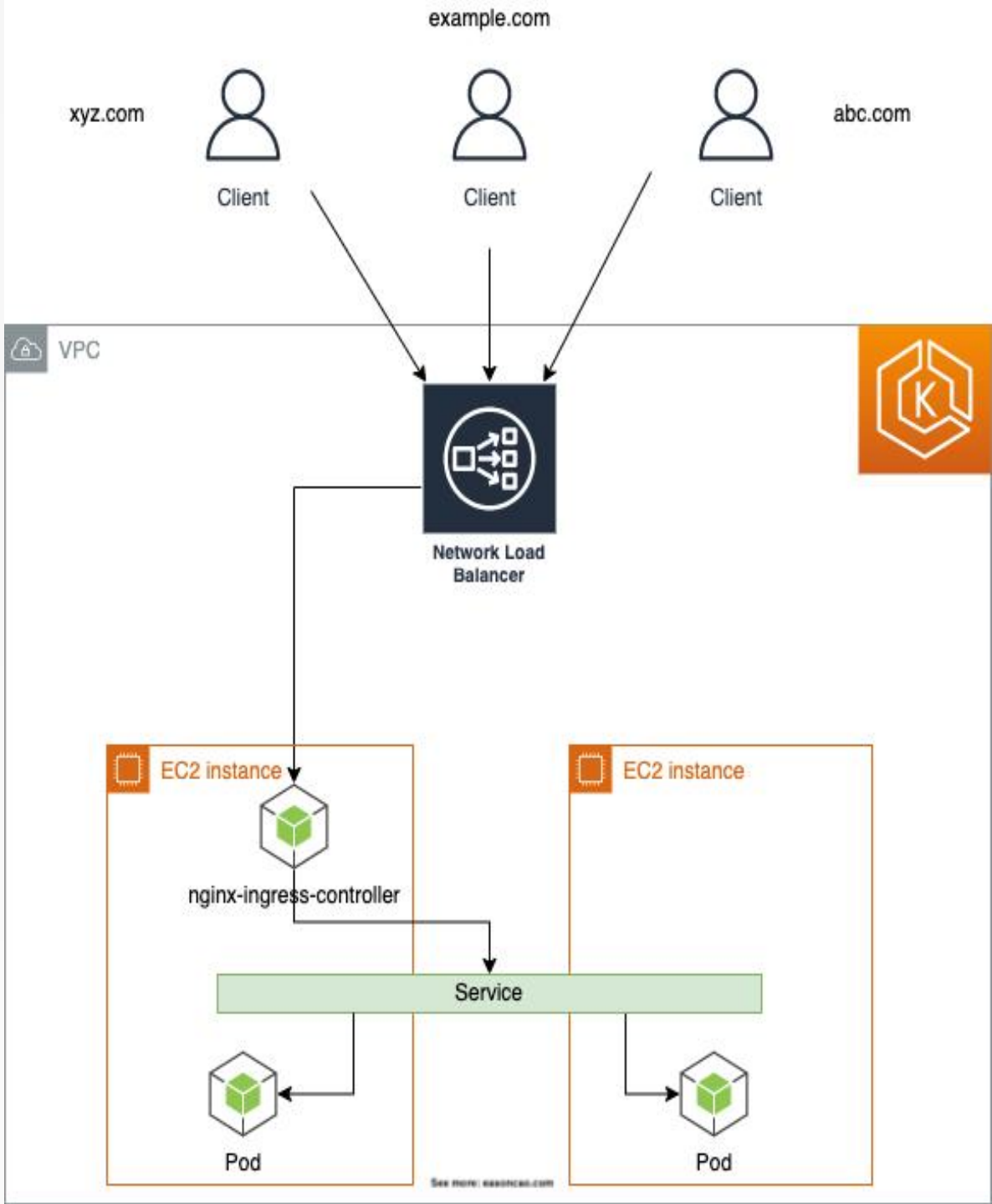
- In Kubernetes, an ingress lets us route traffic from outside the cluster to one or more services inside the cluster.

- Typically, the ingress works as a single point of entry for all incoming traffic. Then, using a set of rules, it forwards all of its traffic to an appropriate service. In turn, that service will send the request to a pod that can actually handle the request.
  - **Kubernetes Ingress Resource:** Kubernetes ingress resource is responsible for storing routing rules in the cluster.
  - **Kubernetes Ingress Controller:** Kubernetes ingress controllers (Nginx/HAProxy etc.) are responsible for routing by accessing the DNS rules applied through ingress resources.


Traffic
Ingress
foo.mydomain.com
mydomain.com/bar
Other
Service
Service
Service
Pod
Pod
Pod
Pod
Pod
Pod
Pod
Pod
Pod
Kubernetes cluster

Ingress Controller:

- Ingress controller is not a native Kubernetes implementation.
- We need to set up an ingress controller for the ingress rules to work. There are several open-source and enterprise ingress controllers available.
- Nginx is one of the widely used ingress controllers.



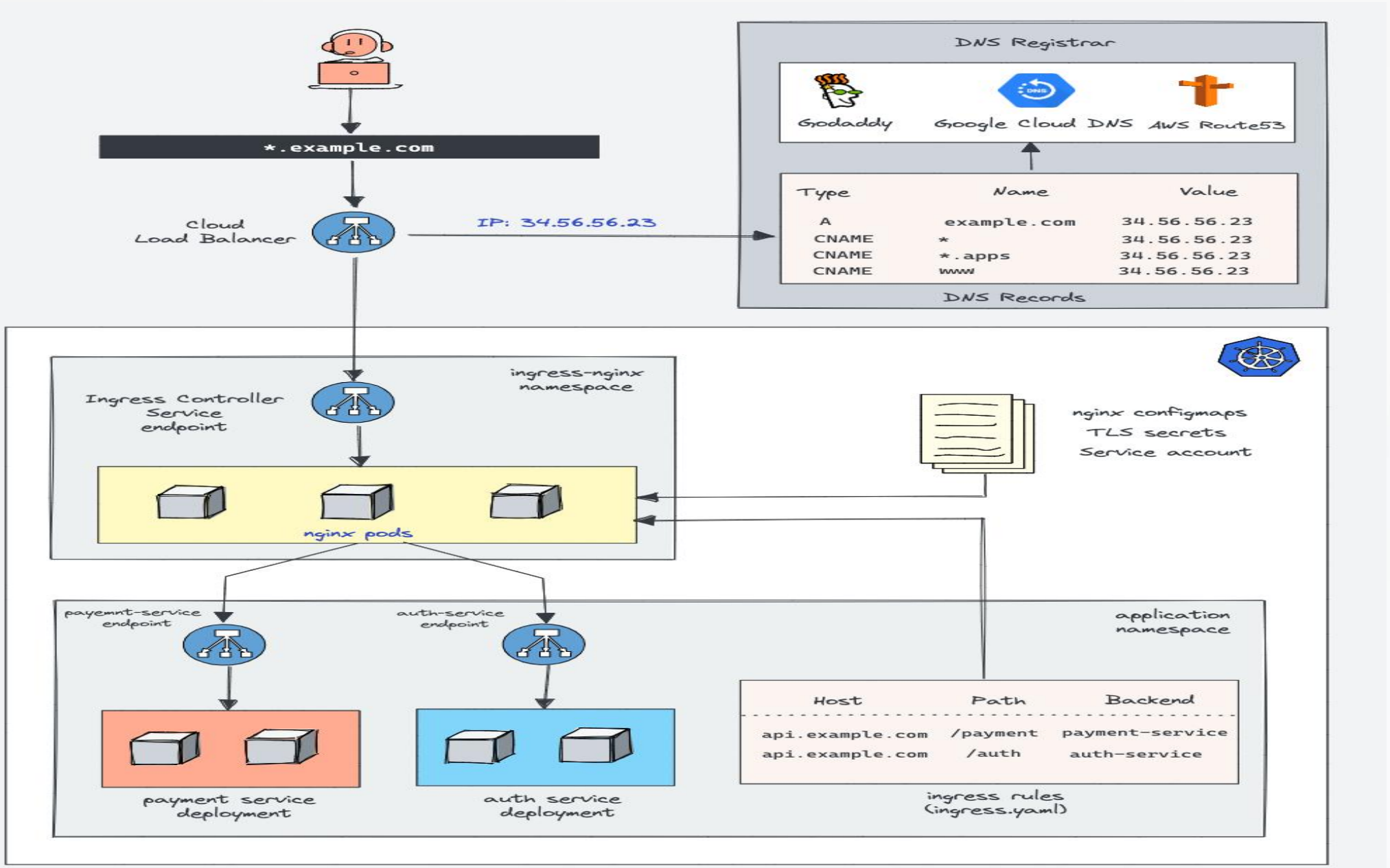
NGINX Ingress Controller

- To use the NGINX Ingress Controller in a Kubernetes cluster, you need to deploy the NGINX Ingress Controller as a deployment in your cluster. You also need to create Ingress resources in your cluster to define the rules for routing traffic to the services.

- The NGINX Ingress Controller runs as a pod within a Kubernetes cluster. It watches for changes to the Ingress resources in the cluster and updates its configuration accordingly. The Nginx controller talks to Kubernetes ingress API to check if there is any rule created for traffic routing.

- When a client sends a request to the NGINX Ingress Controller, it looks up the Ingress rules configured in the cluster to determine which service to route the request to. The NGINX Ingress Controller then forwards the request to the appropriate service and returns the response to the client.

Ingress & Ingress Controller Architecture:



How does NGINX Controller work?

- The *nginx.conf* file inside the Nginx controller pod is a lua template that can talk to Kubernetes ingress API and get the latest values for traffic routing in real-time.
- The Nginx controller talks to Kubernetes ingress API to check if there is any rule created for traffic routing.
- If it finds any ingress rules, the Nginx controller generates a routing configuration inside /etc/nginx/conf.d location inside each nginx pod.
- For each ingress resource you create, Nginx generates a configuration inside /etc/nginx/conf.d location.
- The main /etc/nginx/nginx.conf file contains all the configurations from etc/nginx/conf.d.
- If you update the ingress object with new configurations, the Nginx config gets updated again and does a graceful reload of the configuration.

Components of NGINX Ingress Controller Manifest:

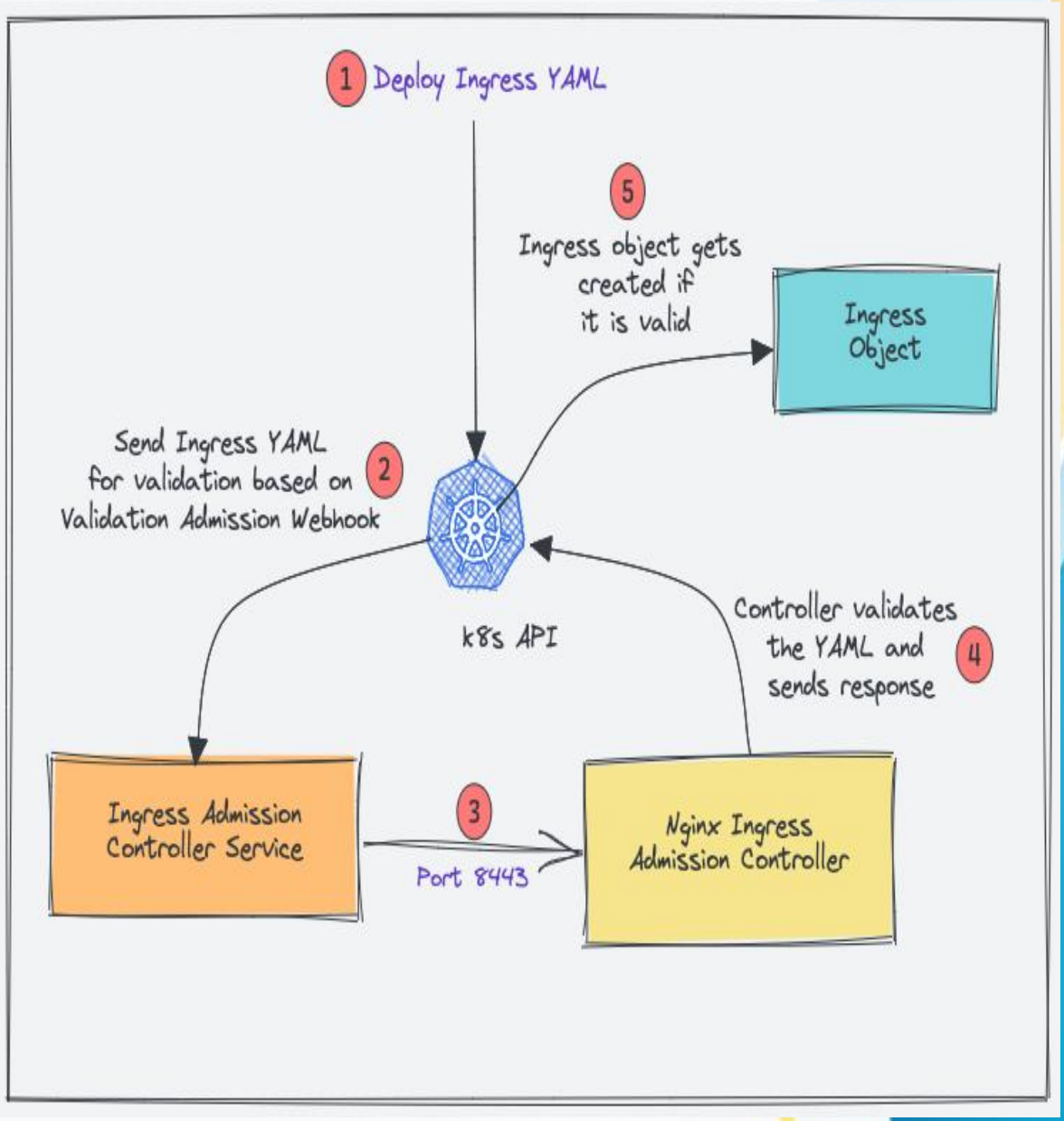
To have a working Nginx controller, we need to deploy the following Kubernetes objects :

- ingress-nginx namespace
- Service account/Roles/ClusterRoles for Nginx admission controller
- Validating webhook Configuration
- Jobs to create/update Webhook CA bundles
- Service account/Roles/ClusterRoles of Nginx controller deployment
- Nginx controller configmap
- Services for nginx controller & admission controller
- Ingress controller deployment

NGINX Admission Controller:

Admission Controller ensures that the ingress object you create has the correct configurations and doesn't break routing rules

- When you deploy an ingress YAML, the Validation admission intercepts the request.
- Kubernetes API then sends the ingress object to the validation admission controller service endpoint based on admission webhook endpoints.
- Service sends the request to the Nginx deployment on port 8443 for validating the ingress object.
- The admission controller then sends a response to the k8s API.
- If it is a valid response, the API will create the ingress object.



More components:

- **ValidatingWebhookConfiguration:** This validates the API object and accepts or rejects it. In this case it will send the object after validation to Admission Controller Service endpoint which will forward it via 8443 to Admission Controller for validation. The ValidatingWebhookConfiguration works only over HTTPS. So it needs a CA bundle.

- **Jobs:** There are two jobs in the controller manifest. One creates a CA bundle for WebhookValidation and the other binds the bundle to the WebhookValidation.

- **ServiceAccounts:** They are created for Admission Controller and Controller Deployment. Roles and clusterRoles are mapped to them using Role and clusterRoleBinding.

- **ConfigMaps:** With this configmap, you can customize the Nginx settings. For example, you can set custom headers and most of the Nginx settings.

- **IngressClass**: This is created and later used in Ingress Resource. This helps the controller read the Ingress rules.

Thank You!