K8s Load balancing

Alessandro Costantini

alessandro.costantini@cnaf.infn.it

Managing network connectivity

- Kubernetes provides several **mechanisms** to manage network connettivity, both internal and external, to handle different scenarios and requirements.
- The most used are:
  - Internal connectivity
    - **ClusterIP**
  - External connectivity
    - **NodePort**
    - **Ingress**
    - **LoadBalancer**

Internal connectivity

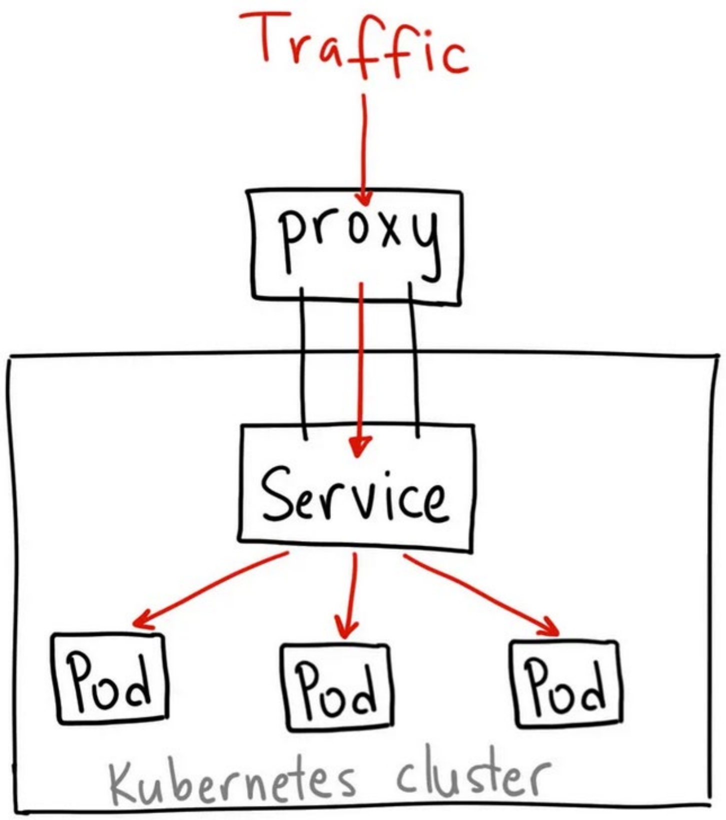
- Refers to distributing traffic within the Kubernetes cluster, typically among pods of the same application or service.
  - Distributing traffic across pods to improve performance and reliability.
  - High availability ensures traffic can still be routed to pods even if some are unavailable.
  - Isolating traffic between different applications or services.

ClusterIP

- **ClusterIP** service type creates an **internal load balancer** that exposes the service to pods within the same cluster.
- **ClusterIP** services do not have a public IP, it has a virtual IP and can only be accessed by pods within the cluster.
- **This IP address** is stable and doesn’t change even if the pods behind the service are rescheduled or replaced.

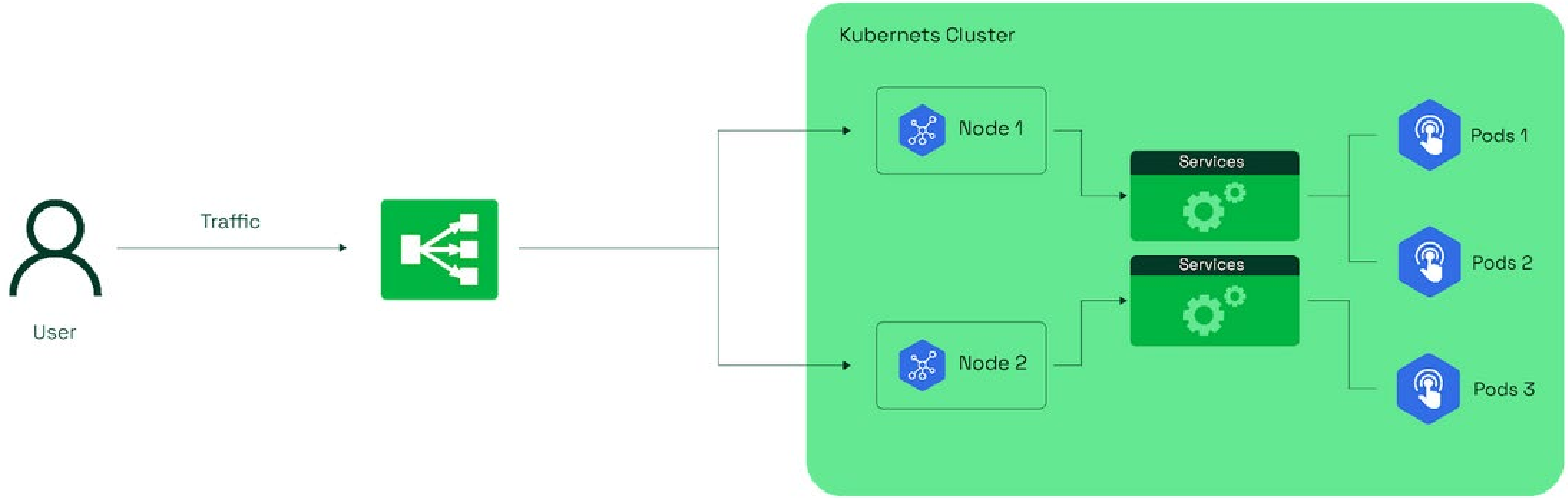
Cluster IP

- Using the **Kubernetes Proxy** we can access the service via the Kubernetes API
- Usage:
  - **Debugging** your services, or connecting to them directly from your laptop for some reason
  - Allowing internal traffic, displaying internal dashboards, etc.



External Connectivity

- Refers to distributing traffic from outside the Kubernetes cluster to appropriate pods within the cluster.

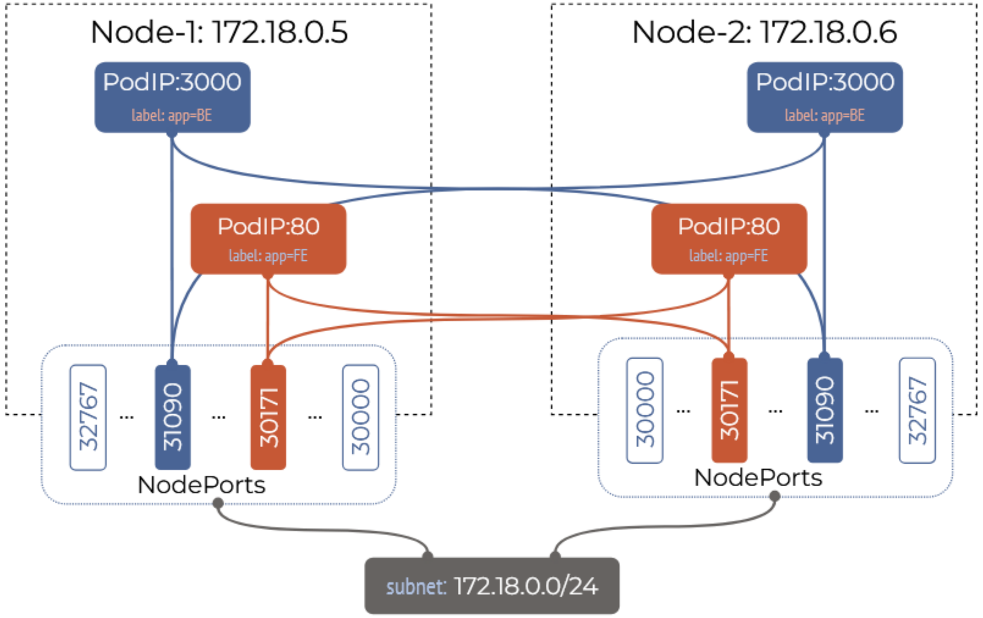


External connectivity

- **NodePort**
- **LoadBalancer**
- **Ingress**

NodePort

- Exposes a specific port on each node in the cluster, allowing access to your service through that port.
- The Kubernetes control plane assigns a port within a specified range (typically 30000-32767).
- Each node then acts as a proxy for the same port number, ensuring consistent service access.



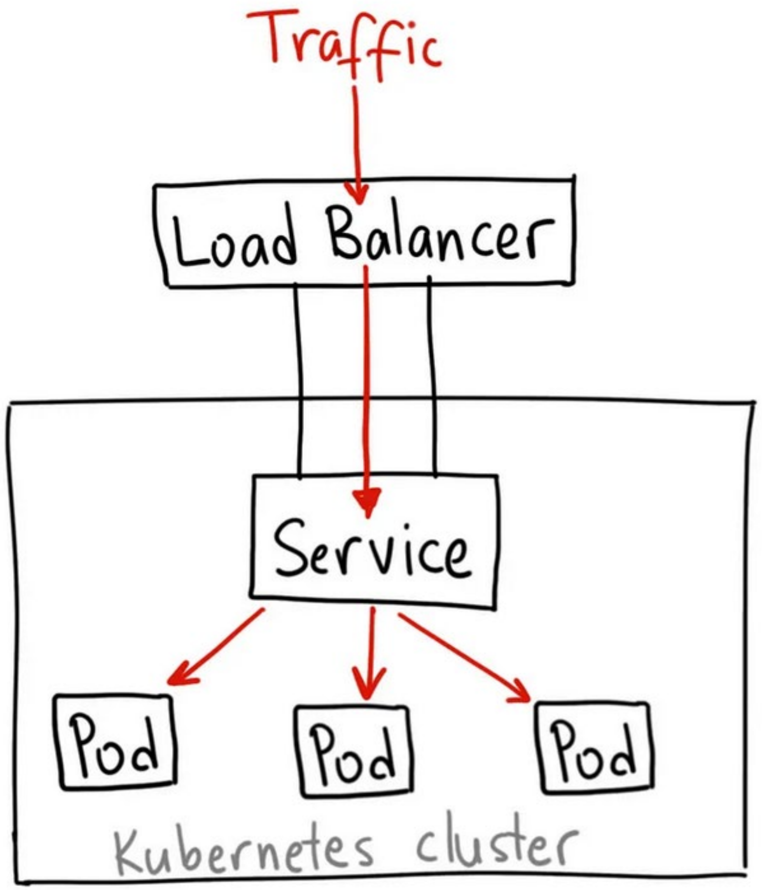
NodePort

- You can only have one service per port
- You can only use ports 30000–32767
- It doesn't do any kind of load balancing, it simply directs traffic

```yaml
apiVersion: v1
kind: Service
metadata:
  name: my-nodeport-service
spec:
  selector:
    app: my-app
  type: NodePort
  ports:
  - name: http
    port: 80
    targetPort: 80
    nodePort: 30036
    protocol: TCP
```

External load balancer

- Provisions an external load balancer, **typically supplied by cloud providers**, to distribute incoming traffic uniformly to the service.
- These services serve as traffic controllers, efficiently directing client requests to the appropriate nodes hosting your pods.



External load balancer

- Used to directly expose a service.
- All traffic on the port you specify will be forwarded to the service.
- There is no filtering, no routing, etc. This means you can send almost any kind of traffic to it, like HTTP, TCP, UDP, Websockets, gRPC, or whatever.

```yaml
apiVersion: v1
kind: Service
metadata:
  name: api-service
spec:
  selector:
    app: api-app
  ports:
  - protocol: TCP
    port: 80
    targetPort: 8080
  type: LoadBalancer
```

External load balancer

- External load balancers **exist outside of the Kubernetes cluster**

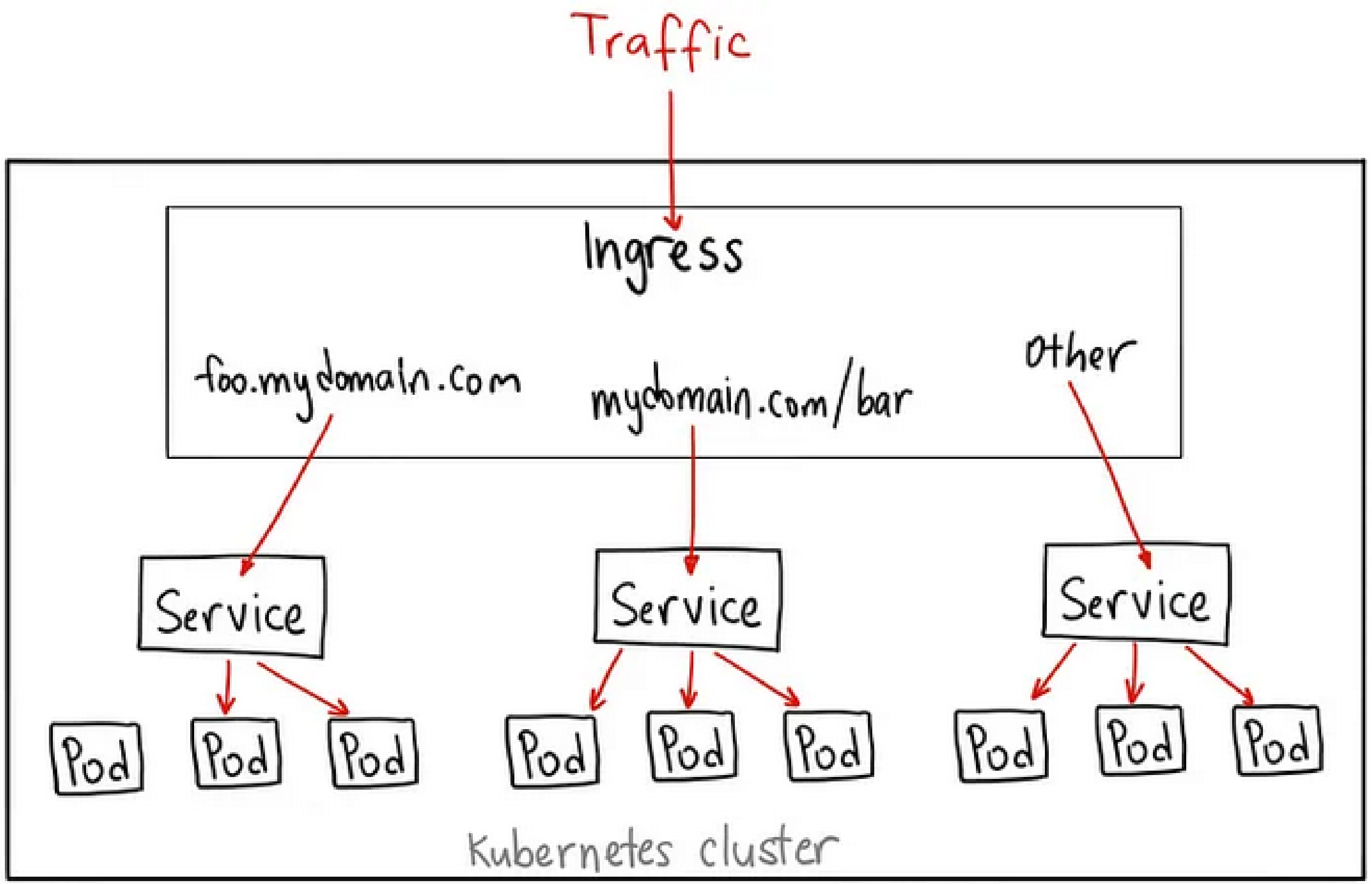
So…

- **the cluster must be running on a provider that supports external load balancers**
- different load balancer providers have their own settings
- are defined per *service*, **they can only route to a single *service***

Ingress

- Ingress is a native Kubernetes resource that exposes HTTP and HTTPS routes from outside the cluster to services within the cluster.
- It relies on rules set in the Ingress resource to control traffic routing.
- Helps on DNS routing.
- Can provide SSL termination and name-based virtual hosting.

Ingress




```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: my-ingress
spec:
  backend:
    serviceName: other
    servicePort: 8080
  rules:
  - host: foo.mydomain.com
    http:
      paths:
      - backend:
          serviceName: foo
          servicePort: 8080
  - host: mydomain.com
    http:
      paths:
      - path: /bar/*
        backend:
          serviceName: bar
          servicePort: 8080
```

Ingress

- Ingress is actually NOT a **type** of service
- act as a “smart router” or entrypoint into the cluster.

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: my-ingress
spec:
  backend:
    serviceName: other
    servicePort: 8080
  rules:
  - host: foo.mydomain.com
    http:
      paths:
      - backend:
          serviceName: foo
          servicePort: 8080
  - host: mydomain.com
    http:
      paths:
      - path: /bar/*
        backend:
          serviceName: bar
          servicePort: 8080
```

Ingress

- An **Ingress** requires an associated controller to manage it.
- Kubernetes provides controllers for most objects like *deployments* and *services*, **it does not include an *ingress controller* by default**.
- The most popular is the **nginx ingress controller** (AWS, GCE also supported and maintained).
- Annotations field used to pass specific configurations into the *ingress controller*.

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ingress-example
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
```

https://kubernetes.io/docs/concepts/services-networking/ingress-controllers/

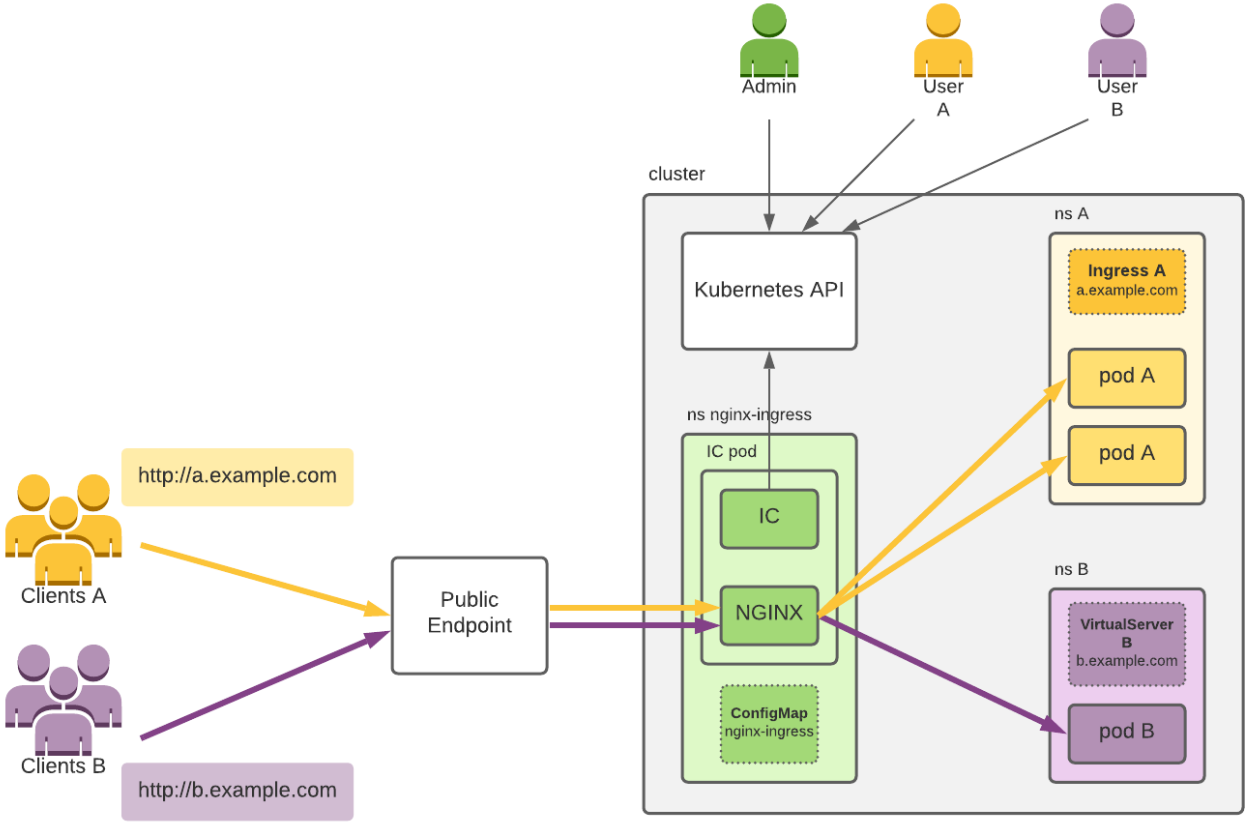
NGINX Ingress Controller

- an Ingress Controller implementation for NGINX and NGINX Plus
- can work with Websocket, TCP and UDP applications.
- supports standard Ingress features such as content-based routing and TLS/SSL termination

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ingress-example
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
```

https://kubernetes.io/docs/concepts/services-networking/ingress-controllers/

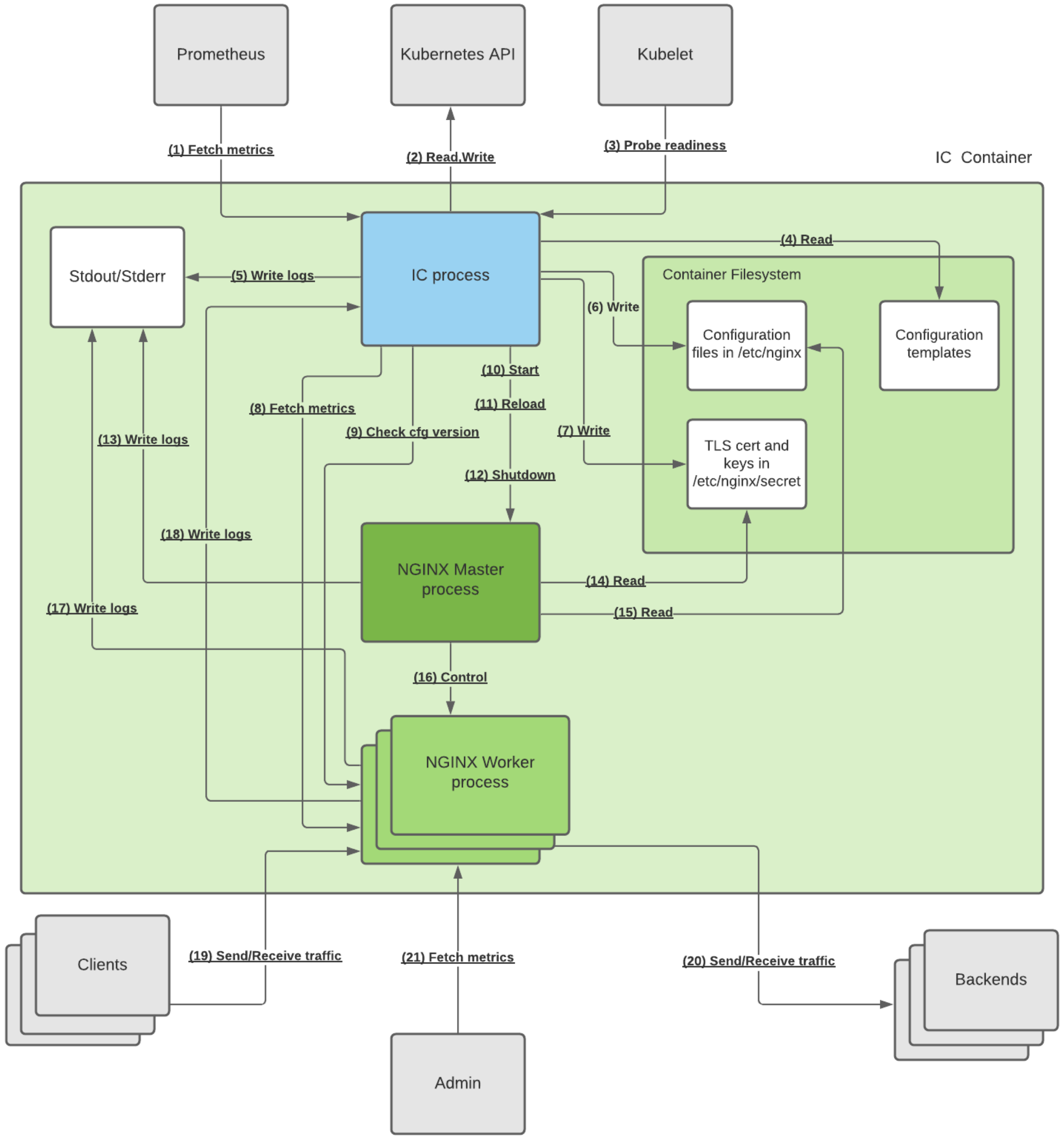
NGINX Ingress Controller



https://docs.nginx.com/nginx-ingress-controller/overview/design/

- *NGINX Ingress Controller*
  - Deployed in a pod with the namespace *nginx-ingress* and configured using the *ConfigMap resource nginx-ingress*.
  - Uses the *Kubernetes API* to get the latest Ingress resources created in the cluster and then configures *NGINX* according to those resources.

NGINX Ingress Controller



- *NGINX Ingress Controller pod*
  - **The *NGINX Ingress Controller process***, which configures NGINX according to Ingress and other resources created in the cluster.
  - **The *NGINX master process***, which controls NGINX worker processes.
  - ***NGINX worker processes***, which handle the client traffic and load balance the traffic to the backend applications.

https://docs.nginx.com/nginx-ingress-controller/overview/design/

Kubernetes services comparison

CLUSTERIP VS NODEPORT VS LOADBALANCER VS INGRESS

More details at tinyurl.com/k8s-service

	ClusterIP Service	NodePort Service	LoadBalancer Service	Ingress + Service
Native K8s Resource	Yes	Yes	Yes, but needs cloud provider load balancer	Yes, but needs ingress controller deployed in cluster
Protocol (OSI Layer)	layer 4	layer 4	layer 4 and below*	layer 7 - http and https only
Allows multiple services per IP	No	No	Yes, but not same port**	Yes
Can expose outside the cluster	No	Yes	Yes (1 service)	Yes (multiple services)

* LoadBalancers are often used in layer 4, but some LoadBalancers support layers 2-3 as well.
For example, https://metallb.universe.tf/concepts/layer2/

** For example, https://kube-vip.io/docs/usage/kubernetes-services/#multiple-services-on-the-same-ip

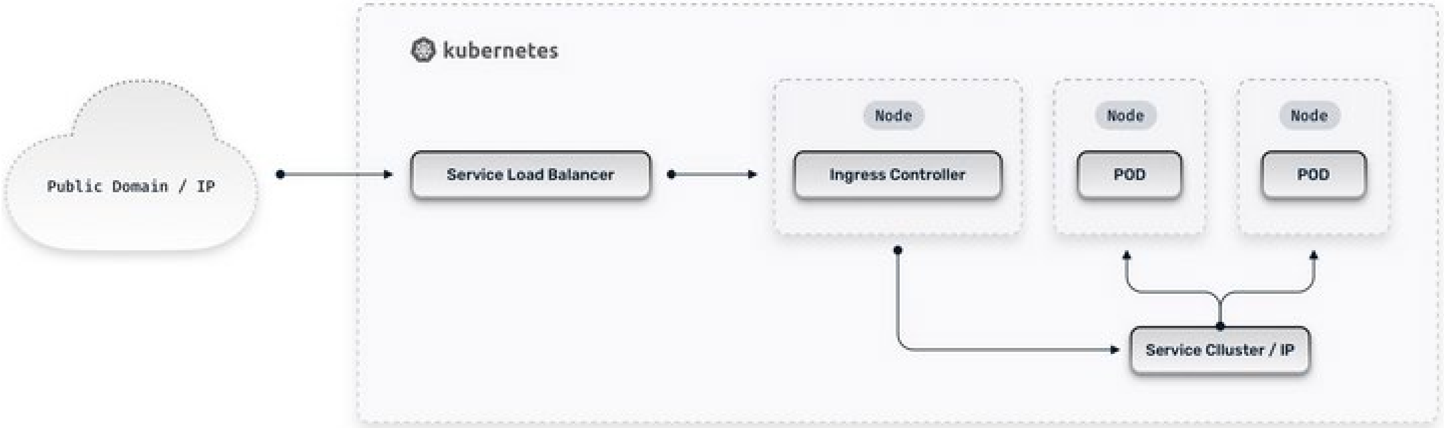
Ingress vs Load balancer

	Ingress	LoadBalancer
Layer	Application layer (L7, HTTP/HTTPS)	Network layer (L4, TCP/UDP)
Use case	Centralized routing for multiple services	Direct exposure for individual services
External IPs	Shares a single external IP	Allocates a unique external IP per service
Features	Advanced routing, SSL termination	Basic load balancing
Cost	More cost-effective (shared IP)	Can be expensive for many services

https://spacelift.io/blog/kubernetes-load-balancer

Ingress + LB

- External load balancers alone aren’t a practical solution for providing the networking capabilities necessary for a K8s environment.
- Kubernetes architecture allows to combine load balancers with an Ingress Controller:
  - Instead of provisioning an external load balancer for every application service that needs external connectivity, we can deploy and configure a single load balancer that targets an Ingress Controller.
  - The Ingress Controller serves as a single entrypoint and can then route traffic to multiple applications in the cluster.



Kubernetes Network Policies

- Mechanism for controlling network traffic flow in Kubernetes clusters
- Each Network Policy targets a group of Pods and sets the **Ingress** (inbound) and **Egress** (outbound) network endpoints those Pods can communicate with.
  - All Pods in a Kubernetes cluster **should be subject to Network Policies** that limit their network interactions to the minimal set of Ingress/Egress targets they require.

There are three different ways to identify target endpoints:

- **Specific Pods** (Pods matching a label are allowed)
- **Specific Namespaces** (all Pods in the namespace are allowed)
- **IP address blocks** (endpoints with an IP address in the block are allowed)

https://kubernetes.io/docs/concepts/services-networking/network-policies/

Allow and Deny

Not setting Network Policies allows all Pods to communicate, which is a potential security risk.

- It is possible to use Network Policies
  - to block all network communications for a Pod
  - to restrict traffic to a specific port range.

Network Policies are additive, so you can have multiple policies targeting a particular Pod

- The sum of the “allow” rules from all the policies will apply
- **Set a default deny policy, then add your allow policies**

Common use cases

- **Ensuring a database can only be accessed by the app it's part of**
  - Databases running in Kubernetes are often intended to be solely accessed by other in-cluster Pods, such as the Pods that run your app's backend.
  - Network Policies allow you to enforce this constraint, preventing other apps from communicating with your database server.
- **Isolating Pods from your cluster's network**
  - Some sensitive Pods might not need to accept any inbound traffic from other Pods in your cluster.
  - Using a Network Policy to block all Ingress traffic to them will tighten your workload's security.
- **Allow specific apps or namespaces to communicate with each other**
  - Kubernetes namespaces are the primary mechanism for separating objects associated with different apps, teams, and environments.
  - You can use Network Policies to network-isolate these resources and achieve stronger multi-tenancy.

Best practices

- Carefully **consider your requirements**. Is a layer 4 load balancer sufficient for your needs, or do you require the option for application layer 7 routing or more advanced features such as SSL termination?
- **Different implementation**, **different features**. Consult the documentation of the solution you are using (Ingress controller, Cloud load balancer).
- Implement **readiness** and **liveness probes** to check the health of your pods, enabling the load balancer to distribute traffic only to healthy instances.
- **Enable connection draining** where supported. Connection draining ensures that existing connections are gracefully handled when a pod or instance is being terminated or scaled.
- **Properly configure Pod autoscaling** to automatically scale the number of pods based on resource utilization or custom metric.
- **Regularly monitor** your system and analyze metrics.

Best practices

- **Apply security best practices**, such as enabling SSL/TLS termination on the load balancer and ensure proper access controls (IAM) are in place to prevent unauthorized access.
- **Simulating failure scenarios** to test your configuration.