KUBERNETES

K8S NETWORK MODEL

Every Pod in a cluster gets its own unique cluster-wide IP address. This means you do not need to explicitly create links between Pods and you almost never need to deal with mapping container ports to host ports. This creates a clean, backwards-compatible model where Pods can be treated much like VMs or physical hosts from the perspectives of port allocation, naming, service discovery, load balancing, application configuration, and migration.

Kubernetes imposes the following fundamental requirements:

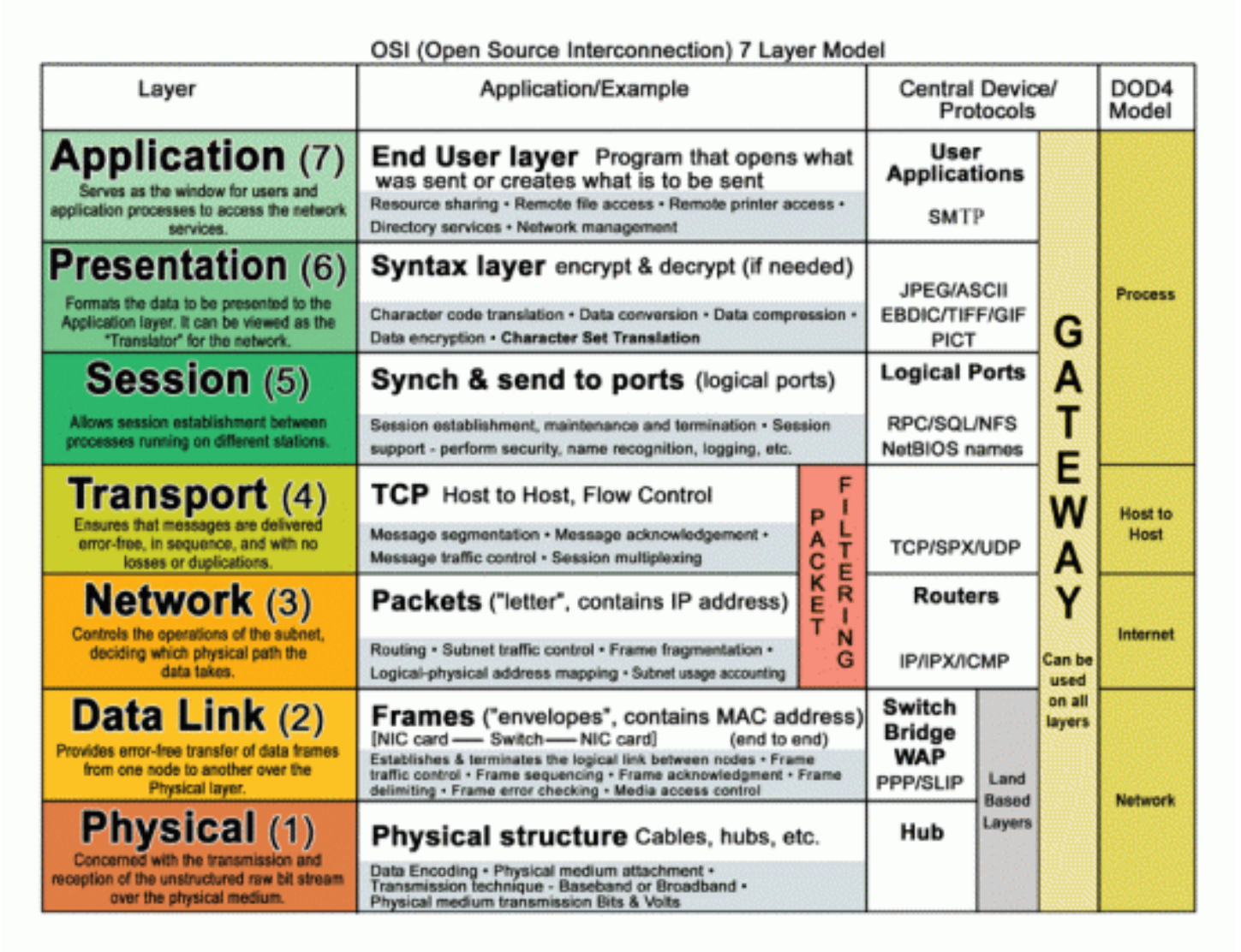
- pods can communicate with all other pods on any other node without NAT
- agents on a node (e.g. system daemons, kubelet) can communicate with all pods on that node

Kubernetes IP addresses exist at the Pod scope - containers within a Pod share their network namespaces (ip addr list) - including their IP address and MAC address. This means that containers within a Pod can all reach each other's ports on localhost. This also means that containers within a Pod must coordinate port usage, but this is no different from processes in a VM. This is called the "IP-per-pod" model.

Kubernetes networking addresses four concerns:

- Containers within a Pod use networking to communicate via loopback.
- Cluster networking provides communication between different Pods.
- The Service API lets you expose an application running in Pods to be reachable from outside your cluster.
- Ingress provides extra functionality specifically for exposing HTTP applications, websites and APIs.
- Gateway API is an add-on that provides an expressive, extensible, and role-oriented family of API kinds for modeling service networking.

OSI LAYERS


OSI (Open Source Interconnection) 7 Layer Model
Layer
Application/Example
Central Device/ Protocols
DOD4 Model
Application (7)
Serves as the window for users and application processes to access the network services.
End User layer Program that opens what was sent or creates what is to be sent
Resource sharing • Remote file access • Remote printer access • Directory services • Network management
User Applications
SMTP
Presentation (6)
Formats the data to be presented to the Application layer. It can be viewed as the "Translator" for the network.
Syntax layer encrypt & decrypt (if needed)
Character code translation • Data conversion • Data compression • Data encryption • Character Set Translation
JPEG/ASCII
EBDIC/TIFF/GIF
PICT
Session (5)
Allows session establishment between processes running on different stations.
Synch & send to ports (logical ports)
Session establishment, maintenance and termination • Session support - perform security, name recognition, logging, etc.
Logical Ports
RPC/SQL/NFS
NetBIOS names
Transport (4)
Ensures that messages are delivered error-free, in sequence, and with no losses or duplications.
TCP Host to Host, Flow Control
Message segmentation • Message acknowledgement • Message traffic control • Session multiplexing
PACKET FILTERING
TCP/SPX/UDP
Network (3)
Controls the operations of the subnet, deciding which physical path the data takes.
Packets ("letter", contains IP address)
Routing • Subnet traffic control • Frame fragmentation • Logical-physical address mapping • Subnet usage accounting
Routers
IP/IPX/ICMP
Data Link (2)
Provides error-free transfer of data frames from one node to another over the Physical layer.
Frames ("envelopes", contains MAC address)
[NIC card — Switch — NIC card] (end to end)
Establishes & terminates the logical link between nodes • Frame traffic control • Frame sequencing • Frame acknowledgment • Frame delimiting • Frame error checking • Media access control
Switch
Bridge
WAP
PPP/SLIP
Land Based Layers
Physical (1)
Concerned with the transmission and reception of the unstructured raw bit stream over the physical medium.
Physical structure Cables, hubs, etc.
Data Encoding • Physical medium attachment • Transmission technique - Baseband or Broadband • Physical medium transmission Bits & Volts
Hub
GATEWAY
Can be used on all layers
Process
Host to Host
Internet
Network

NETWORK ADDONS

- **ACI** provides integrated container networking and network security with Cisco ACI.
- **Antrea** operates at Layer 3/4 to provide networking and security services for Kubernetes, leveraging Open vSwitch as the networking data plane. Antrea is a CNCF project at the Sandbox level.
- **Calico** is a networking and network policy provider. Calico supports a flexible set of networking options so you can choose the most efficient option for your situation, including non-overlay and overlay networks, with or without BGP. Calico uses the same engine to enforce network policy for hosts, pods, and (if using Istio & Envoy) applications at the service mesh layer.
- **Cilium** is a networking, observability, and security solution with an eBPF-based data plane. Cilium provides a simple flat Layer 3 network with the ability to span multiple clusters in either a native routing or overlay/encapsulation mode. Cilium is a CNCF project at the Graduated level.
- **Flannel** is an overlay network provider that can be used with Kubernetes.
- **Gateway API** is an open source project managed by the SIG Network community and provides an expressive, extensible, and role-oriented API for modeling service networking.
- **Knitter** is a plugin to support multiple network interfaces in a Kubernetes pod.

NETWORK ADDONS

Multus is a Multi plugin for multiple network support in Kubernetes to support all CNI plugins (e.g. Calico, Cilium, Contiv, Flannel), in addition to SRIOV, DPDK, OVS-DPDK and VPP based workloads in Kubernetes.

- **OVN**-Kubernetes is a networking provider for Kubernetes based on OVN (Open Virtual Network), a virtual networking implementation that came out of the Open vSwitch (OVS) project. OVN-Kubernetes provides an overlay based networking implementation for Kubernetes, including an OVS based implementation of load balancing and network policy.
- **Nodus** is an OVN based CNI controller plugin to provide cloud native based Service function chaining(SFC).
- **NSX-T** Container Plug-in (NCP) provides integration between VMware NSX-T and container orchestrators such as Kubernetes, as well as integration between NSX-T and container-based CaaS/PaaS platforms such as Pivotal Container Service (PKS) and OpenShift.
- **Nuage** is an SDN platform that provides policy-based networking between Kubernetes Pods and non-Kubernetes environments with visibility and security monitoring.
- **Romana** is a Layer 3 networking solution for pod networks that also supports the NetworkPolicy API.
- **Weave** Net provides networking and network policy, will carry on working on both sides of a network partition, and does not require an external database.

KUBE PROXY

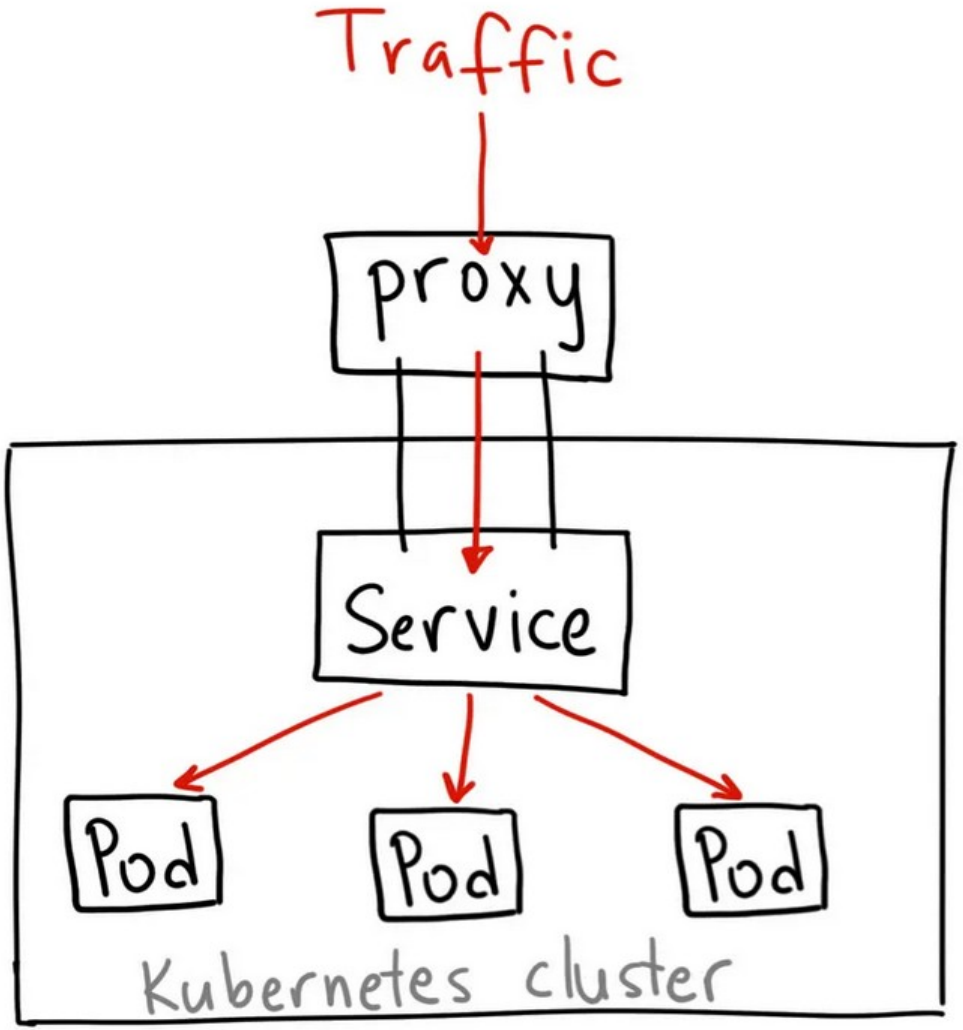
NAMESPACE KUBE-SYSTEM

- Cilium
- Coredns
- **Kube-proxy**

kube-proxy is a network proxy that runs on each node in your cluster, implementing part of the Kubernetes Service concept.

kube-proxy maintains network rules on nodes. These network rules allow network communication to your Pods from network sessions inside or outside of your cluster.

kube-proxy uses the operating system packet filtering layer if there is one and it's available. Otherwise, kube-proxy forwards the traffic itself.



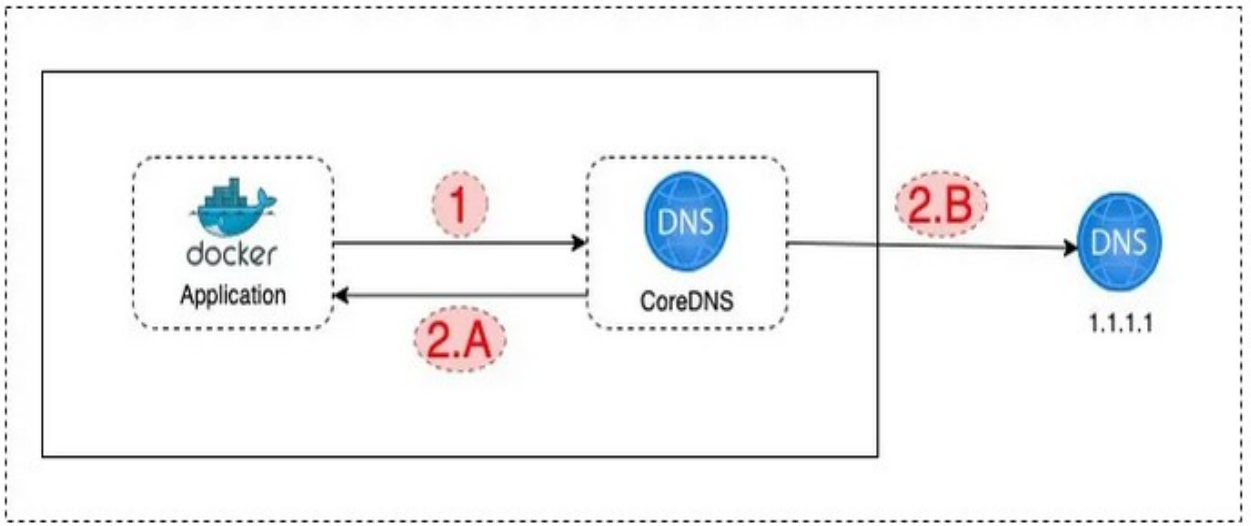
COREDNS

NAMESPACE KUBE-SYSTEM

- Cilium
- **Coredns**
- Kube-proxy

CoreDNS is a flexible, extensible DNS server that can serve as the Kubernetes cluster DNS. Like Kubernetes, the CoreDNS project is hosted by the CNCF.

You can use CoreDNS instead of kube-dns in your cluster by replacing kube-dns in an existing deployment, or by using tools like kubeadm that will deploy and upgrade the cluster for you

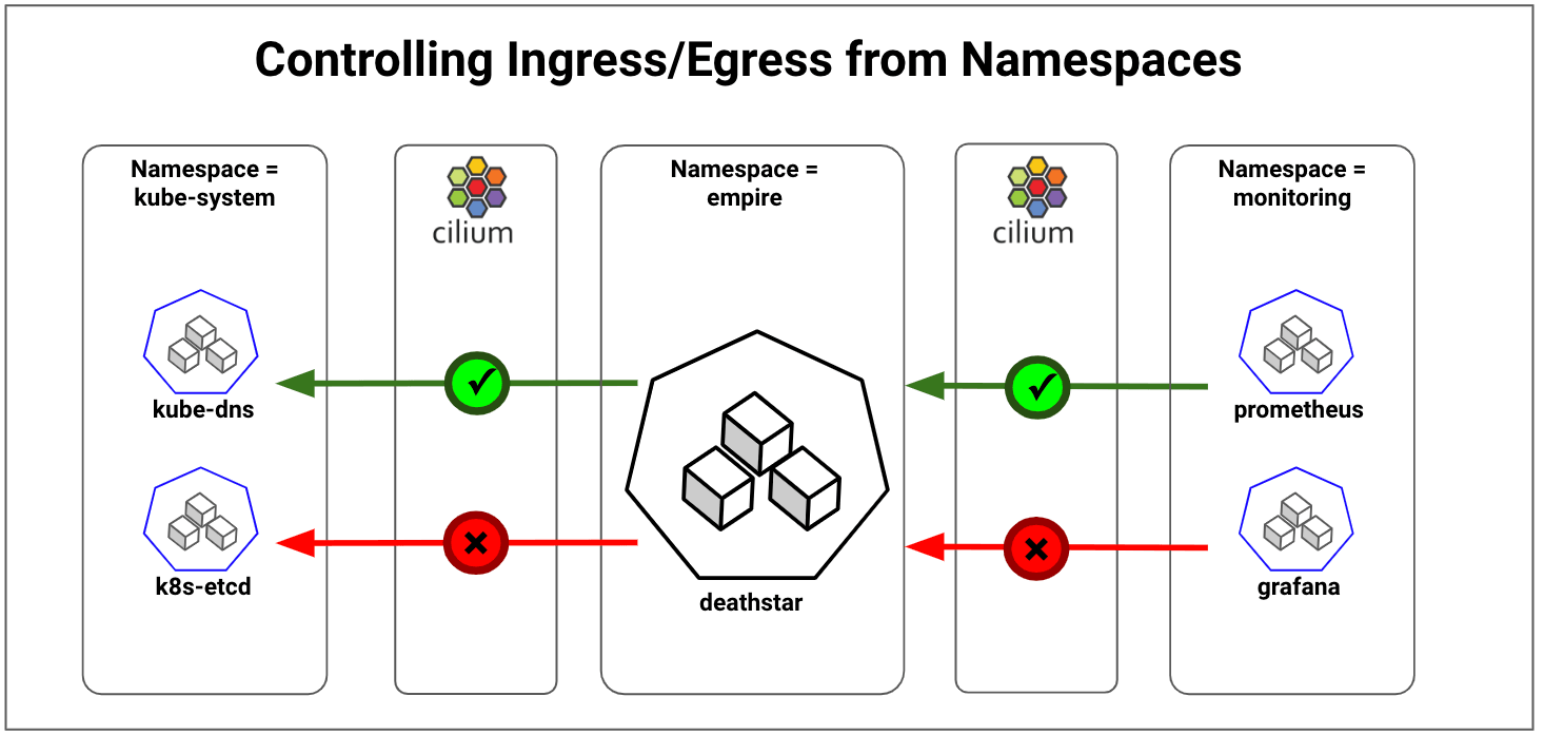


CILIUM

NAMESPACE KUBE-SYSTEM

- **Cilium**
- Coredns
- Kube-proxy

Cilium enables you to secure the network connectivity between application services deployed in Kubernetes



K8S OBJECTS - NETWORKING

SERVICE: An abstract way to expose an application running on a set of Pods as a network service. With Kubernetes you don't need to modify your application to use an unfamiliar service discovery mechanism. Kubernetes gives Pods their own IP addresses and a single DNS name for a set of Pods, and can load-balance across them. Kubernetes Service types allow you to specify what kind of Service you want. The available type values and their behaviors are:

ClusterIP (Exposes the Service on a cluster-internal IP).
NodePort (Exposes the Service on each Node's IP at a static port)
LoadBalancer (Exposes the Service externally using an external load balancer)

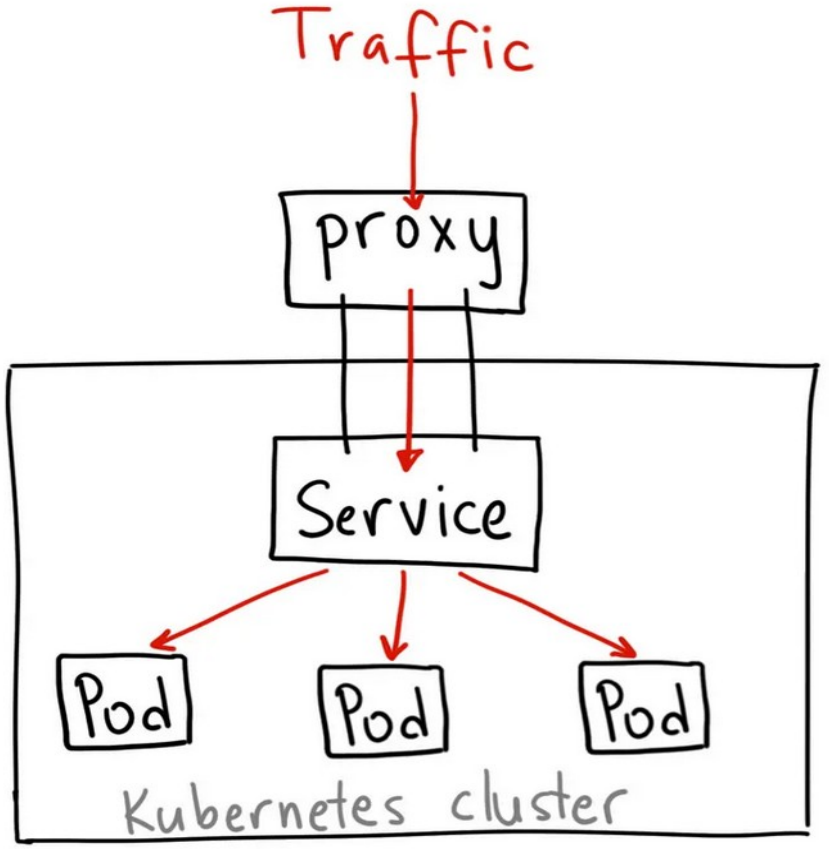
NETWORK POLICIES: If you want to control traffic flow at the IP address or port level (OSI layer 3 or 4), then you might consider using Kubernetes NetworkPolicies for particular applications in your cluster. There are two sorts of isolation for a pod: isolation for egress, and isolation for ingress.

INGRESS: it may provide load balancing, SSL termination and name-based virtual hosting. Ingress exposes HTTP and HTTPS routes from outside the cluster to services within the cluster. Traffic routing is controlled by rules defined on the Ingress resource.

CLUSTERIP

A ClusterIP service is the default Kubernetes service. It gives you a service inside your cluster that other apps inside your cluster can access. There is no external access.

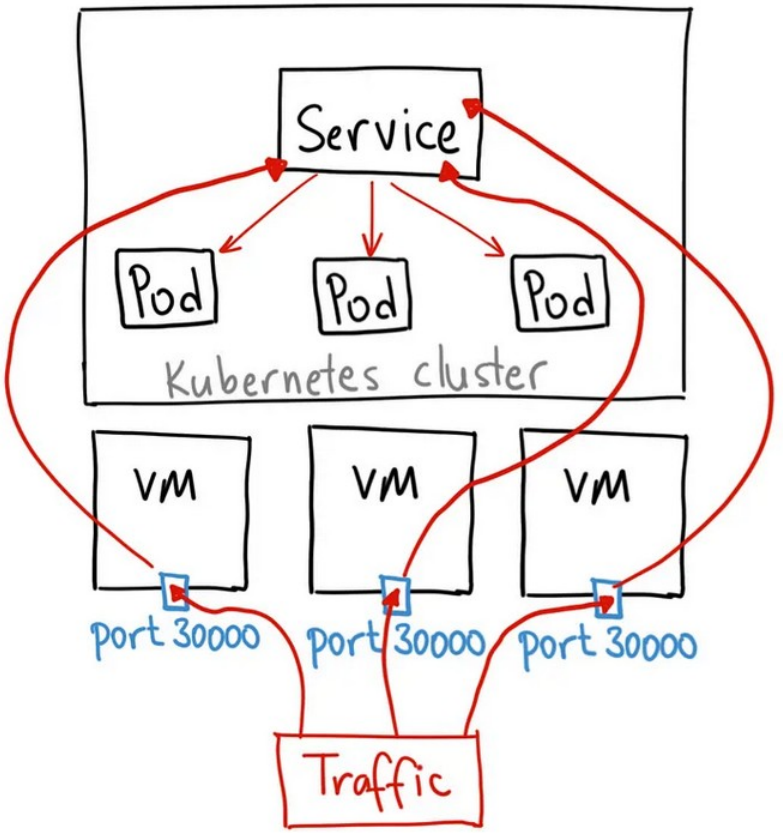
```
apiVersion: v1
kind: Service
metadata:
  name: my-internal-service
spec:
  selector:
    app: my-app
  type: ClusterIP
  ports:
  - name: http
    port: 80
    targetPort: 80
    protocol: TCP
```



NODEPORT

A NodePort service is the most primitive way to get external traffic directly to your service. NodePort, as the name implies, opens a specific port on all the Nodes (the VMs), and any traffic that is sent to this port is forwarded to the service.

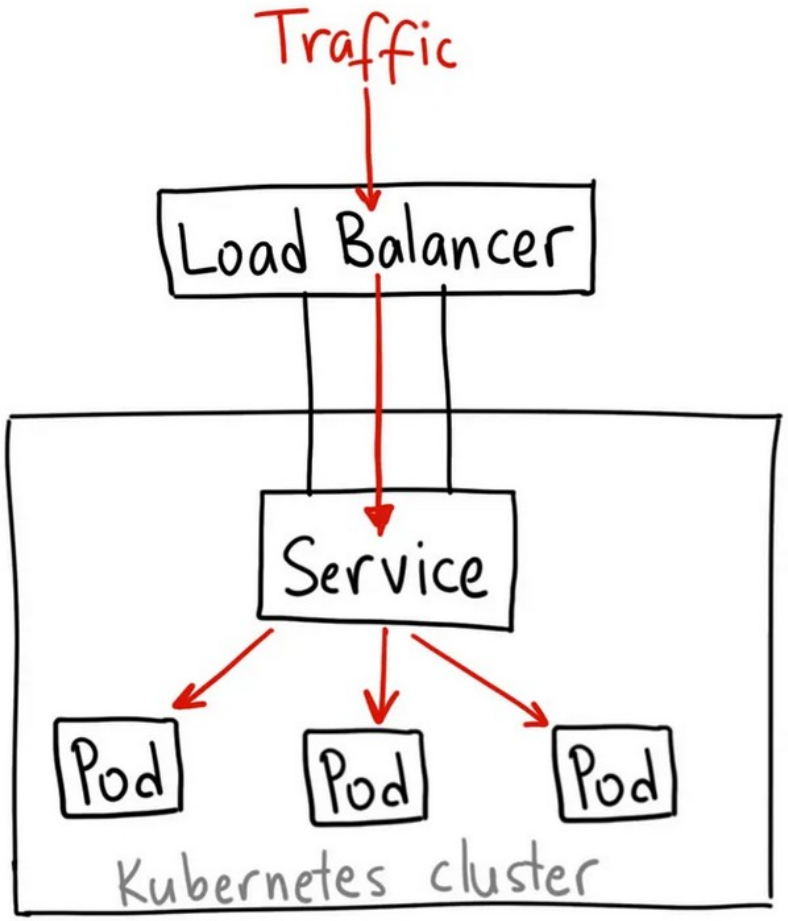
```
apiVersion: v1
kind: Service
metadata:
  name: my-nodeport-service
spec:
  selector:
    app: my-app
  type: NodePort
  ports:
  - name: http
    port: 80
    targetPort: 80
    nodePort: 30036
    protocol: TCP
```



LOADBALANCER

A LoadBalancer service is the standard way to expose a service to the internet. If you want to directly expose a service, this is the default method. 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.

The big downside is that each service you expose with a LoadBalancer will get its own IP address, and you have to pay for a LoadBalancer per exposed service, which can get expensive!



INGRESS

Unlike all the above examples, Ingress is actually NOT a type of service. Instead, it sits in front of multiple services and act as a “smart router” or entrypoint into your cluster.

You can do a lot of different things with an Ingress, and there are many types of Ingress controllers that have different capabilities.

Nginx ingress controller will spin up a Load Balancer for you. This will let you do both path based and subdomain based routing to backend services. For example, you can send everything on foo.yourdomain.com to the foo service, and everything under the yourdomain.com/bar/ path to the bar service.