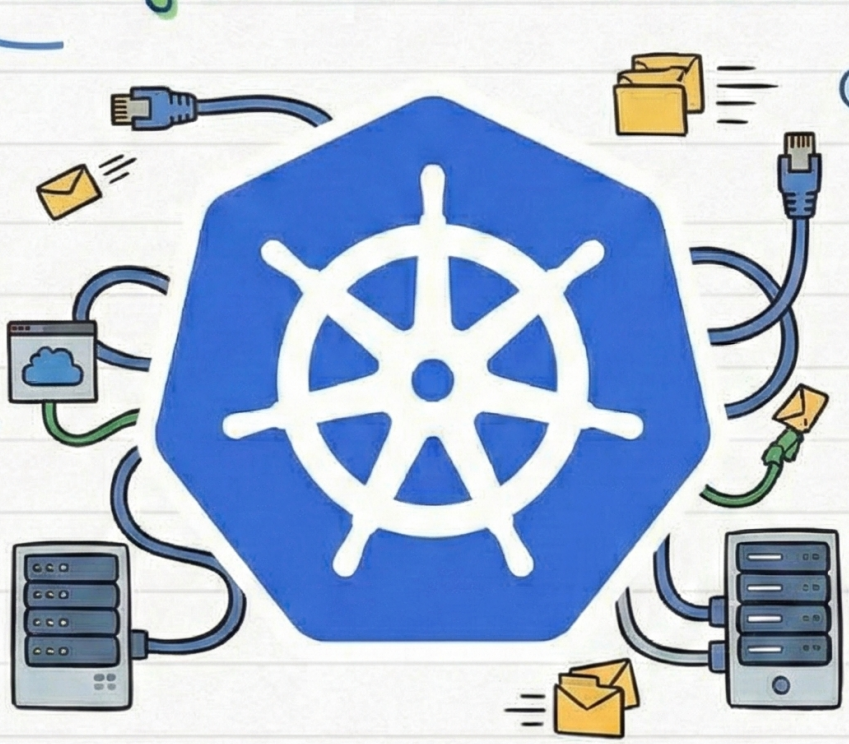
Kubernetes Simplified – Part 2



Handwritten By:
Vaibhav Matkar

Service → stable network endpoint used for communication.

service components

① clusterIP – default servicetype → make service reachability within cluster only. i-e exposes service to internal IP's [communication within cluster]

```yaml
apiVersion: v1
kind: Service
metadata:
  labels:
    k8s-app: kube-dns
    kubernetes.io/cluster-service: "true"
    kubernetes.io/name: CoreDNS
  name: kube-dns
  namespace: kube-system
spec:
  clusterIP: 10.96.0.10
  ports:
  - name: dns
    port: 53
    protocol: UDP
    targetPort: 53
  - name: dns-tcp
    port: 53
    protocol: TCP
    targetPort: 53
  selector:
    k8s-app: kube-dns
  type: ClusterIP
```

② Nodeport – Expose service outside the cluster by opening port on every node. [communication outside cluster]

```yaml
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  type: NodePort
  selector:
    app.kubernetes.io/name: MyApp
  ports:
    - port: 80
      # By default and for convenience, the `targetPort` is set to
      # the same value as the `port` field.
      targetPort: 80
      # Optional field
      # By default and for convenience, the Kubernetes control plane
      # will allocate a port from a range (default: 30000-32767)
      nodePort: 30007
```

③ Externalname → just maps service name to external DNS

```yaml
apiVersion: v1
kind: Service
metadata:
  name: my-service
  namespace: prod
spec:
  type: ExternalName
  externalName: my.database.example.com
```

④ loadbalancer → created by cloud provider exposes service to internet with public ip / DNS

```yaml
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
    app.kubernetes.io/name: MyApp
  ports:
    - protocol: TCP
      port: 80
      targetPort: 9376
  clusterIP: 10.0.171.239
  type: LoadBalancer
status:
  loadBalancer:
    ingress:
    - ip: 192.0.2.127
```

#Network communication in kubernetes

① container to container → via local host as belongs to same network namespace

ii) pod to pod within same Node → Each pod has its own ip so communicate vai IP's using CNI

iii) pod to pod across different Nodes (same cluster)
→ CNI plugin setup routing/ overlay network across nodes to communicate

Kubernetes Network Security & Access control

① Network policies → rules that restricts traffic inside cluster

② RBAC → granular permissions for users & services

③ CNI → plugin based interface allows 3rd party network providers to give additional network security

Ingress in kubernetes

↳ App layer (7th)

↓

mgmt of incoming traffic to cluster using adv routing & rules for HTTP/HTTPS

↓

Functions

① Load Balancing
↓
distribute traffic to different instances of service.

② Routing
↓
distribute traffic to multiple services correctly using routing rules

③ Access control
↓
prevents un-autho-access to any service based on ip.

④ No need of TLS certification for every service instead use in ingress

Encr → Decry

Working of Nodeport service

Allows to expose container/application to outside world

i> create Nodeport service in k8 manifest file

mention → Target port → container listening
→ port to be exposed to outside world
ie. Nodeport
↓
manually / auto [30,000 - 32767]

ii> kubernetes maps Target port to Nodeport

iii> To access application outside kubernetes cluster → http://node-ip/nodeport

kubeproxy

- kubeproxy → Agent installed → as daemonset
  on every node → Linux process
  ↓
  middleware between service & pod

- function → monitor changes in service and endpoints [pod ip's] → create Network rules
  ↓
  decides which request will go to which pod

→ pods are ephemeral → service provides

- Load Balancing between pods
- Stable ip to pods
- DNS name to pods

Working of kubeproxy

① Any new service / endpoints added → API server inform this to kubeproxy

② kubeproxy create NAT rule for this changes. i.e. map service ip to pod ip → when request come to service it is redirected to pod

NAT [Network Address translation]

Rule that forwards ip address/port (1st) to ip address/port (2nd)

- **SNAT**: Internal private IP → public IP
- **DNAT**: redirect external request to appropriate destination

kube proxy modes

User space mode

- Use iptables
- Traffic
- iptables ← redirect rule [@kernel space]
- kube-proxy local port ← contains service to pod rule [@user space]
- forwarded respective pods

Latency & Unpopular

iptables mode

- uses iptables
- Traffic
- iptables contains service to pod mapping rule added by kube proxy
- forwarded respective pods

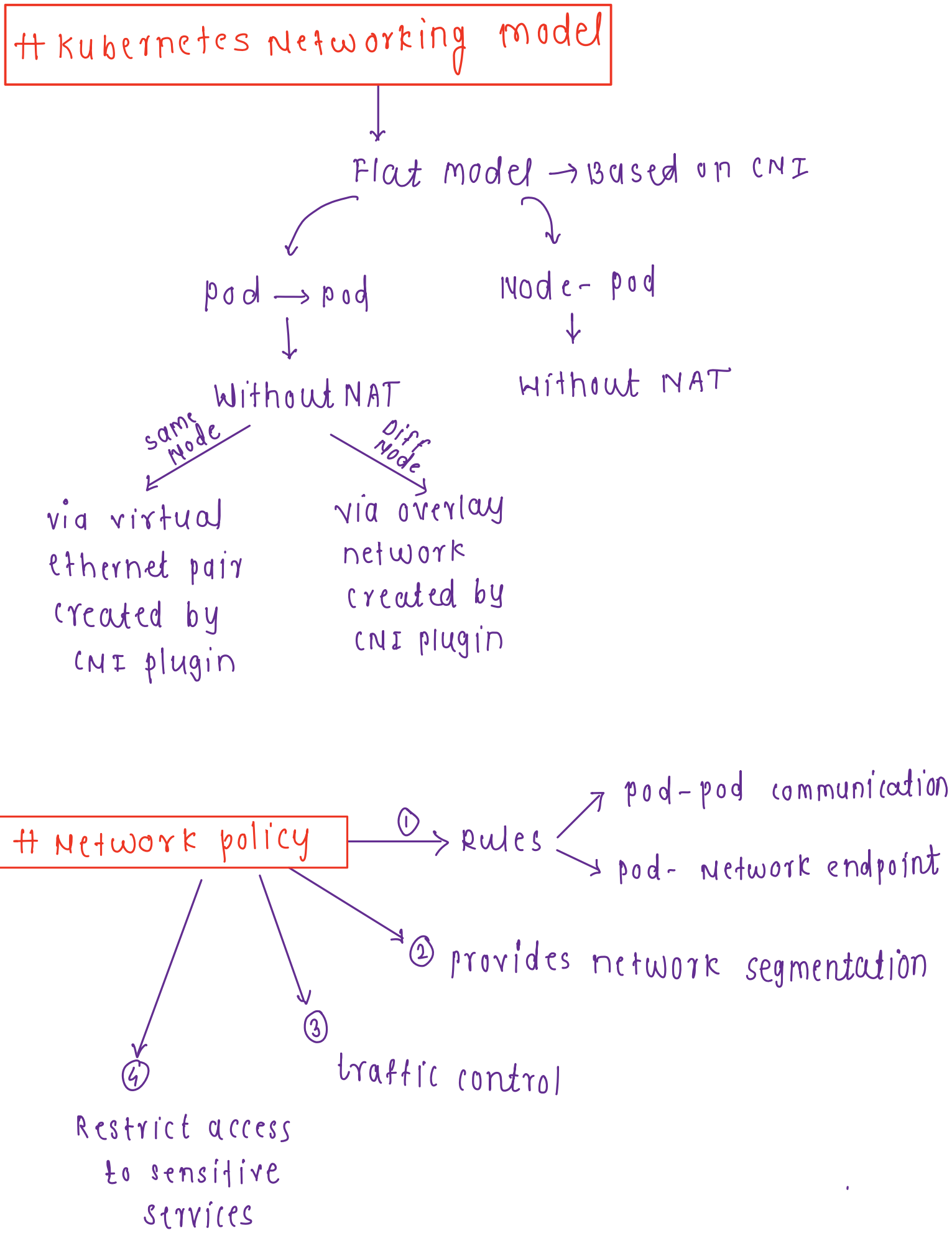
• Default & popular

Ipvs mode

Ip virtual server

- mapping rule is inserted into Ipvs & not iptables by kube proxy
- optimized lookup algorithm for consistent performance
- Adv. Load balancing algorithms

eg. Round robin, Least connection hashing approach


Kubernetes Networking model
Flat model → Based on CNI
Pod → Pod
Node - Pod
Without NAT
Without NAT
same Node
Diff Node
via virtual ethernet pair created by CNI plugin
via overlay network created by CNI plugin
Network policy
① Rules
Pod-Pod communication
Pod - Network endpoint
② provides network segmentation
③ traffic control
④ Restrict access to sensitive services

kubernetes DNS

↳ [Internal DNS] for service discovery

- pod → DNS A record form → pod-ip-addr.namespace.pod.cluster.local
- service → DNS. A record → service-name. namespace.svc.cluster.local

challenges of implementing k8's network policies.

① Larger clusters ⇒ more policy complexity

② Policy conflicts ⇒ more than 1 policy for same pod

③ performance ↓ ⇒ cluster large consumes more resources during policy evaluation leaving few resources.

troubleshooting network connectivity issues in k8's cluster

① verify CNI configuration → directory → /etc/cni/net.d/

② check pod label & selector → incorrect network policy applied to right pod

③ Look for policy conflicts → nx policy → single pod

④ check CNI log files → vary → /var/log/CNI-plug/cnil

Egress Network policy

→ pods are allowed to send traffic outside

Ingress Network policy

→ pods are allowed for incoming traffic

- podselector → select pod matching label

```
podselectors:
    matchLabels:
      app: demo
```

- namespaceselector → select all pod matching label

```
namespaceselector:
    matchLabels:
      app: demo
```

- ipBlock → allow traffic from specific CIDR range

```
ipBlock:
    cidr: <ip>/24
```

- from: [] → rules used for ingress policy
  ↳ default

- to: [] → rules used for egress policy
  ↳ default

#Container Network Interface [CNI]

↳ plugin system that decides Networking in kubernetes.

- Flannel → overlay Network provider
- calico → provides Secure network connectivity

Working of flannel

- Flannel create flat Network
- Assign subnet to each Node

↓

Each pod get ip from subnet

- Flanneld agent maintains routing table mapping subnets to node IP's.
- communication
  - pod-pod (same Node) → via linux bridge [No encapsulation]
  - pod-pod (Across Node) → via VXLAN encapsulation by flanneld of one node & decapsulation by flanneld of other Node.

Calico CNI

↳ open source networking & network security solution

• Calico Architecture

① Birds → BGP routing daemon that distribute routes from Felix to BGP peers [nodes]

② Felix → manages routes/ACL'S & other configurations on each host [Node]

③ confd → Tool that monitors changes in calico configuratn in datastore.

④ IPAM [IP Address mgmt] → Assign ip's to pods using IP Pool resources

⑤ Typha → cache between datastore & felix instance

⑥ Datastore → stores calico related configurations

• Working of calico

- New pod —calls→ calico → Assign unique ip to pod & add pod to linux interface

- Each Node has CIDR range → calico adds route to node's kernel routing table.

- calico allows for Network policies for pod communication.

- calico uses pure layer 3 routing → packet transfers from Node to Node without the encapsulation.

Uses BGP [Border Gateway protocol]

Nodes shares routes with each other, so each node knows which node has which pod.

calico Networking Modes

① IPIP mode → package encapsulation for inter-node communication.

② Direct mode → direct routing without encapsulation

③ VXLAN mode → package encapsulation for inter-node communication via VXLAN interface

④ pod packet Encryption mode → uses WireGuard tunnels & encrypt & transmit pod traffic betwn Nodes.

service discovery in K8's

- service registry → central repository where each service registers network locations/capabilities/metadata
- service discovery mechanism → Allows services to dynamically discover & locate other services. based on service registry.

- Working

① service registration → unique DNS Name & IP address is assigned to pod

② DNS resolution → DNS resolution by other services.

③ Load balancing → during multiple requests for service, K8 automatically load balances traffic to available pod belong to service.

④ service updates → if pods associated with services changes due to scalability → K8 updates service's DNS record for set of available endpoints.

NFS → protocol which allows to mount storage device as local drive on container

- Benefits
  - use existing storage volume
  - persistency
  - used to share data between containers
  - simultaneous mounting → NFS can be mounted on multiple nodes → R|W at same time from multiple nodes.

MTLS in kubernetes

↳ Mutual Transport Layer Security.

↓

- Zero trust mechanism → Both client & server verify each other using certificates and communicate

↓ This is handled by

- Istio
- Linkerd
- consul