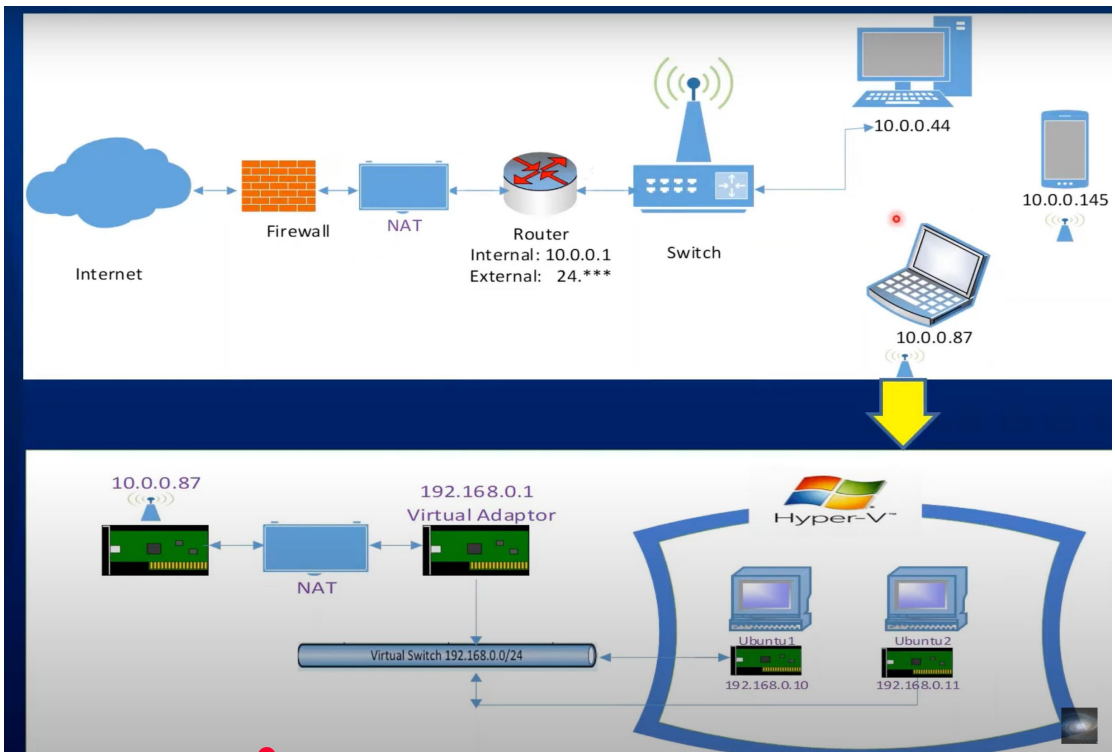
NEW YORK INSTITUTE OF TECHNOLOGY
College of Engineering and Computing Sciences
Department of Computer Science Vancouver Campus

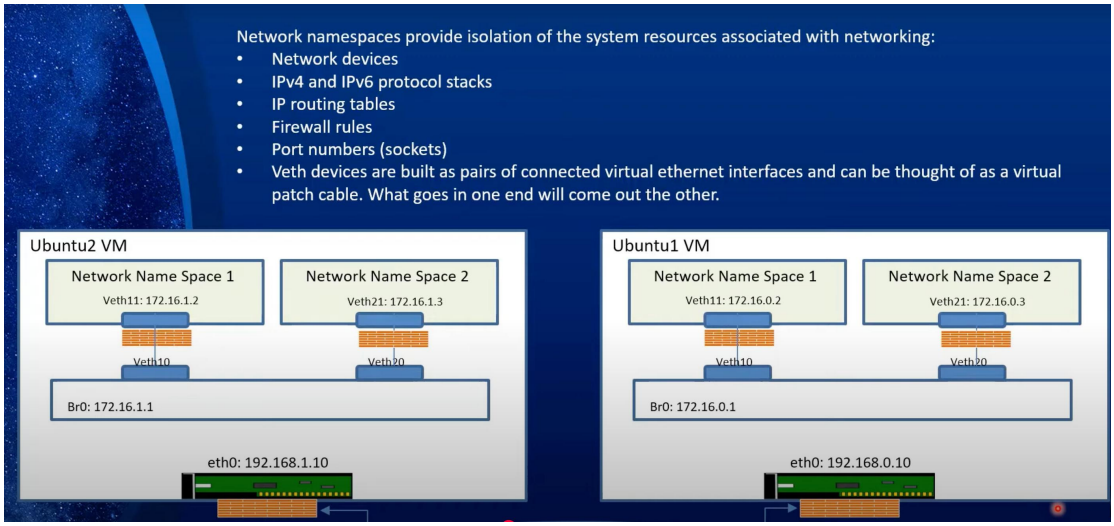
Data Structures - 1339481-zhijun-jiang CSCI 615 Research Presentation

Virtual Network in k8s and the security measure implemented

Virtual Network in the containization with virtual router ans switches



Linux name space to mimic the docker containers. It provides isolation and conmunicate through veth bridge



```
#!/bin/bash -e

#!bash
NS1="NS1"
NS2="NS2"
NODE_IP="192.168.0.10"
BRIDGE_SUBNET="172.16.0.0/24"
BRIDGE_IP="172.16.0.1"
IP1="172.16.0.2"
IP2="172.16.0.3"
TO_NODE_IP="192.168.0.11"
TO_BRIDGE_SUBNET="172.16.1.0/24"
TO_BRIDGE_IP="172.16.1.1"
TO_IP1="172.16.1.2"
TO_IP2="172.16.1.3"
```

```
echo "Creating the namespaces"
sudo ip netns add $NS1
sudo ip netns add $NS2
ip netns show
```

```
echo "Creating the veth pairs"
sudo ip link add veth10 type veth peer name
veth11
sudo ip link add veth20 type veth peer name
veth21
ip link show type veth
#ip link show veth11
#ip link show veth20
```

```
echo "Adding the veth pairs to the namespaces"
sudo ip link set veth11 netns $NS1
sudo ip link set veth21 netns $NS2
```

```
echo "Configuring the interfaces in the network namespaces with IP address"
sudo ip netns exec $NS1 ip addr add $IP1/24 dev veth11
sudo ip netns exec $NS2 ip addr add $IP2/24 dev veth21
```

```
echo "Enabling the interfaces inside the network namespaces"
sudo ip netns exec $NS1 ip link set dev veth11 up
sudo ip netns exec $NS2 ip link set dev veth21 up
```

```
echo "Creating the bridge"
sudo ip link add br0 type bridge
ip link show type bridge
ip link show br0
#sudo ip link delete br0
```

```
echo "Adding the network namespaces interfaces to the bridge"
sudo ip link set dev veth10 master br0
sudo ip link set dev veth20 master br0
```

```
echo "Assigning the IP address to the bridge"
sudo ip addr add $BRIDGE_IP/24 dev br0
```

```
echo "Enabling the bridge"
sudo ip link set dev br0 up
```

```
echo "Enabling the interfaces connected to the bridge"
sudo ip link set dev veth10 up
sudo ip link set dev veth20 up
```

```
echo "Setting the loopback interfaces in the network namespaces"
```

```
sudo ip netns exec $NS1 ip link set lo up
sudo ip netns exec $NS2 ip link set lo up
sudo ip netns exec $NS1 ip a
sudo ip netns exec $NS2 ip a
```

```
echo "Setting the default route in the network
namespaces"
sudo ip netns exec $NS1 ip route add default via
$BRIDGE_IP dev veth11
sudo ip netns exec $NS2 ip route add default via
$BRIDGE_IP dev veth21
```

```
# ------------------- Step 3 Specific Setup -----
---------------- #
```

```
echo "Setting the route on the node to reach the
network namespaces on the other node"
sudo ip route add $TO_BRIDGE_SUBNET via
$TO_NODE_IP dev eth0
```

```
echo "Enables IP forwarding on the node"
sudo sysctl -w net.ipv4.ip_forward=1
```

```
#-------------------Tests----------------------
-------
```

```
#Ping adaptor attached to NS1
sudo ip netns exec $NS1 ping -W 1 -c 2 172.16.0.2
```

```
#Ping the bridge
sudo ip netns exec $NS1 ping -W 1 -c 2 172.16.0.1
```

```
#Ping the adaptor of the second container
sudo ip netns exec $NS1 ping -W 1 -c 2 172.16.0.3
```

```
#Ping the other server (Ubuntu2)
```

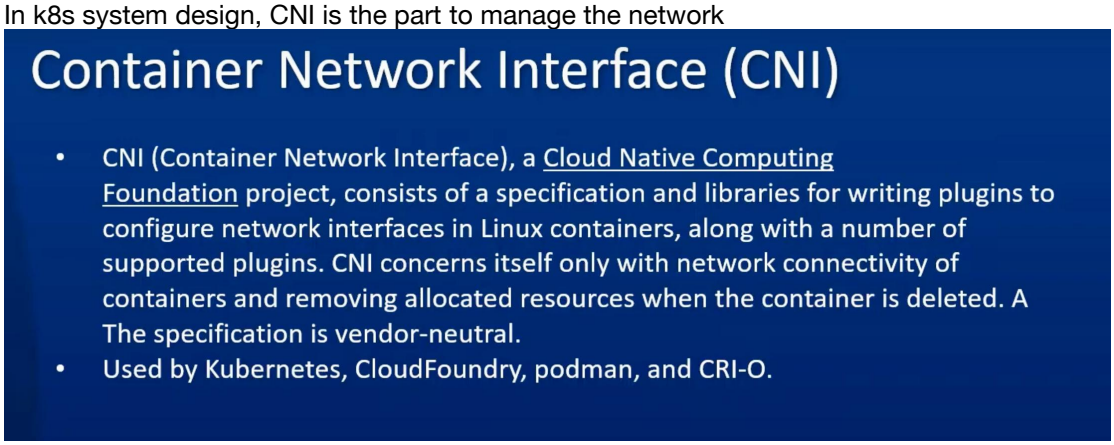
```
sudo ip netns exec $NS1 ping -W 1 -c 2
192.168.0.11
```

```
#Ping the bridge on "Ubuntu2" server
sudo ip netns exec $NS1 ping -W 1 -c 2 172.16.1.1
```

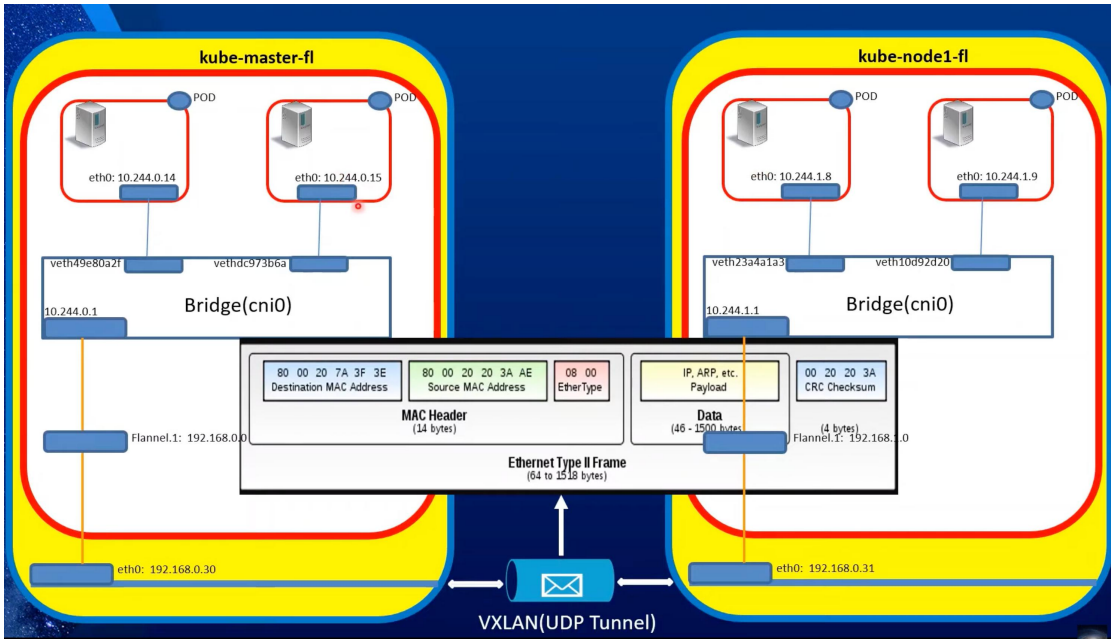
```
#Ping the first container on "Ubuntu2"
sudo ip netns exec $NS1 ping -W 1 -c 2 172.16.1.2
```

```
#Ping the second container on "Ubuntu2"
sudo ip netns exec $NS1 ping -W 1 -c 10
172.16.1.3
```

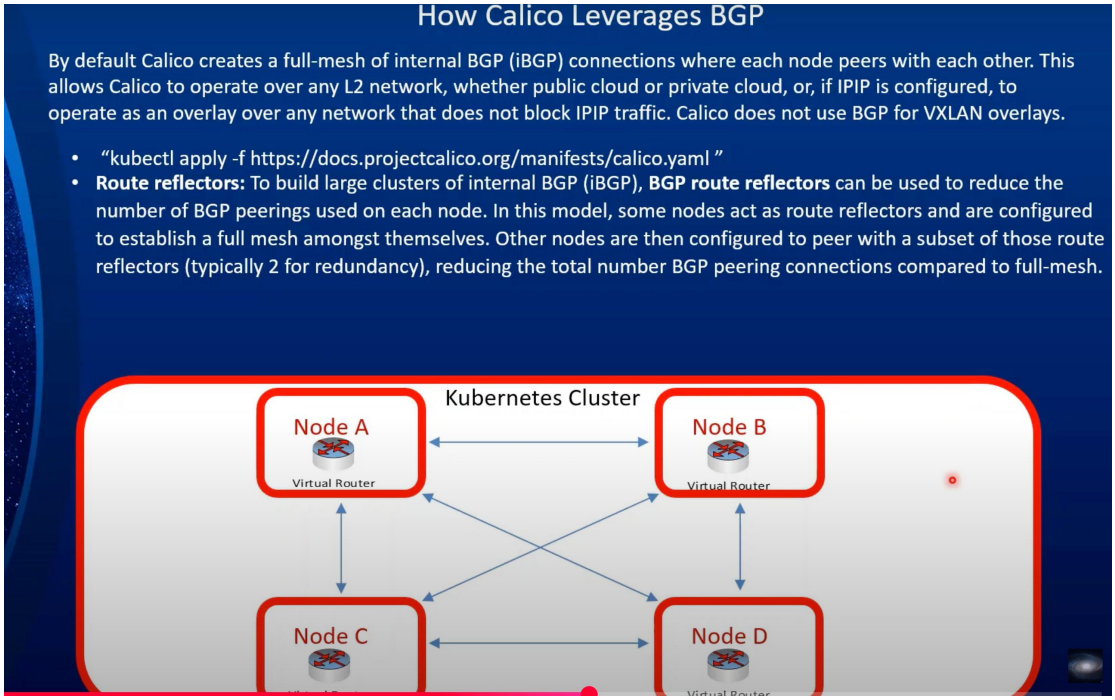
In k8s system design, CNI is the part to manage the network



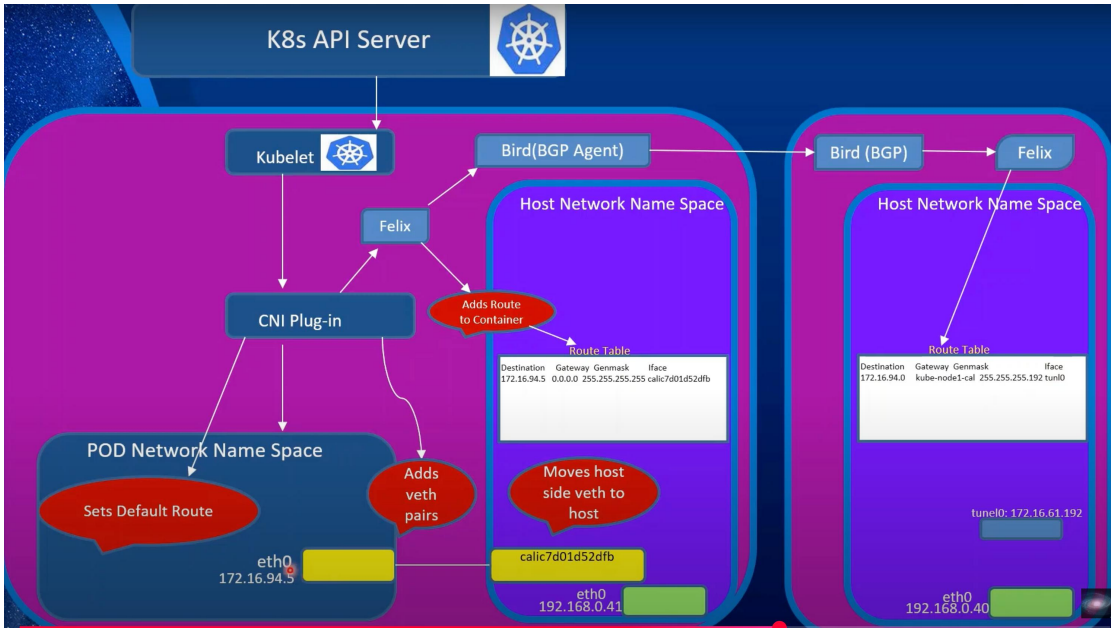
pods communication among the same nodes and different nodes.
same node through the verth and bridge
different nodes through the VXLAN Tunnel



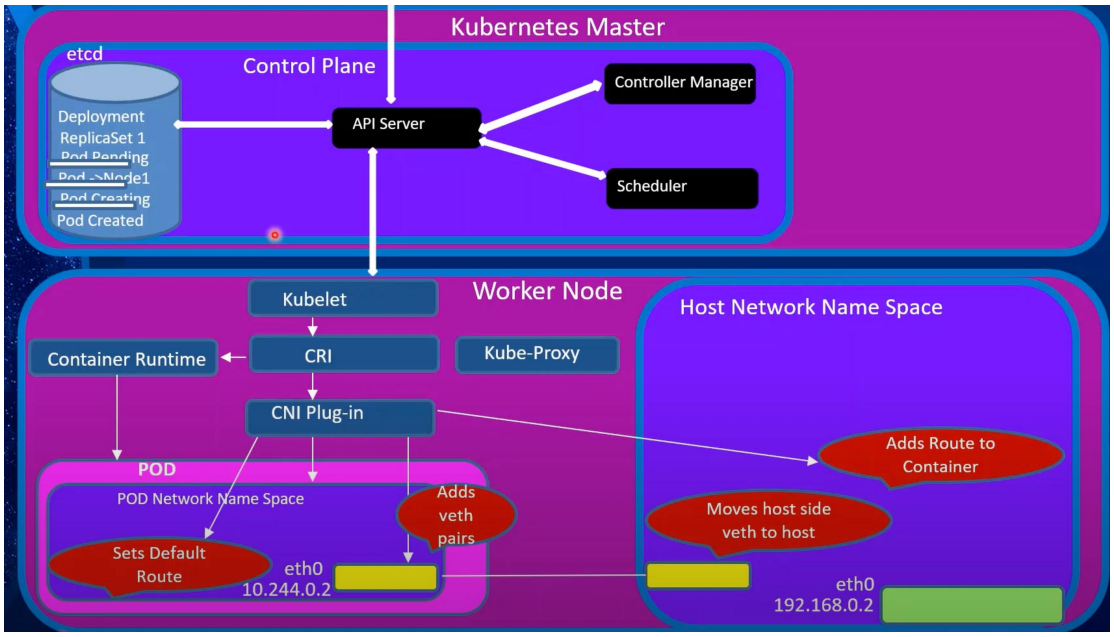
Another CNI implementation is Calico



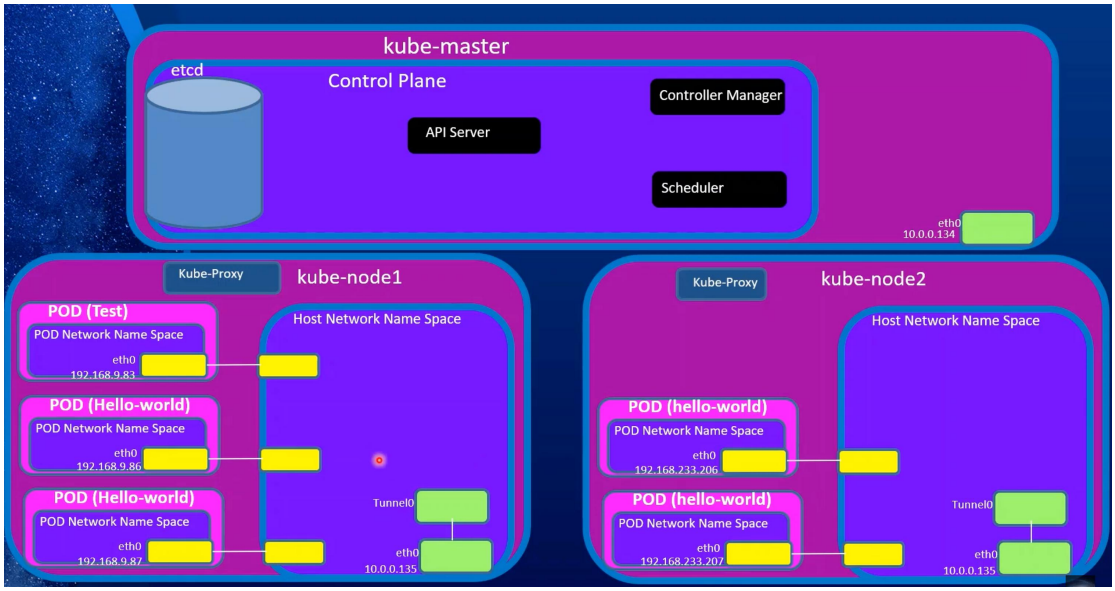
How Calico CNI propagate the pod network



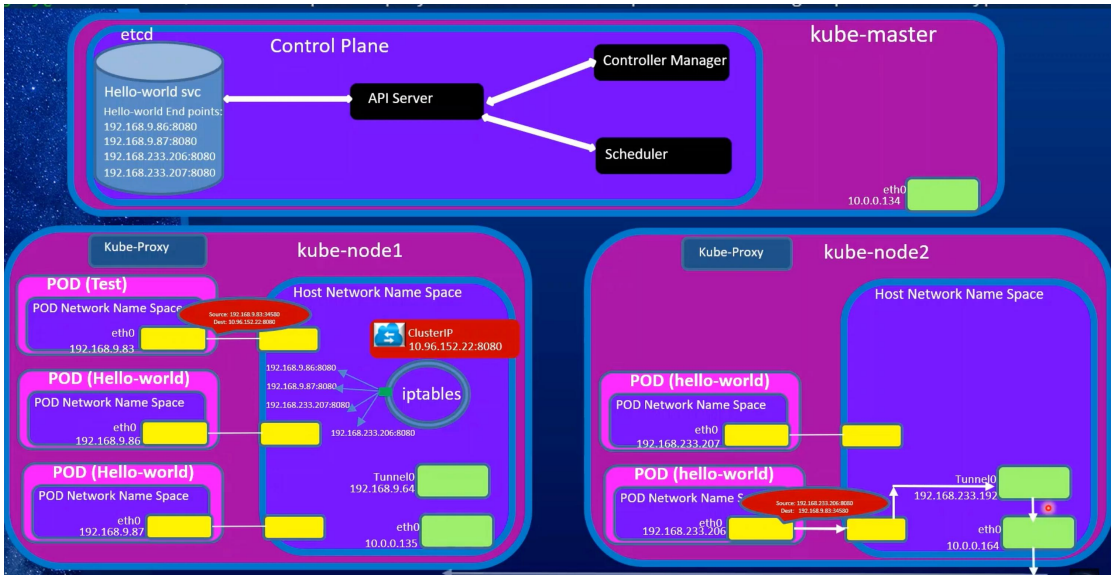
How service works



Pod can communicate with each other at the same or different node through the pod IP but what id pod get restarted which can happens very often



In cluster service leverage iptables to find the pod endpoints



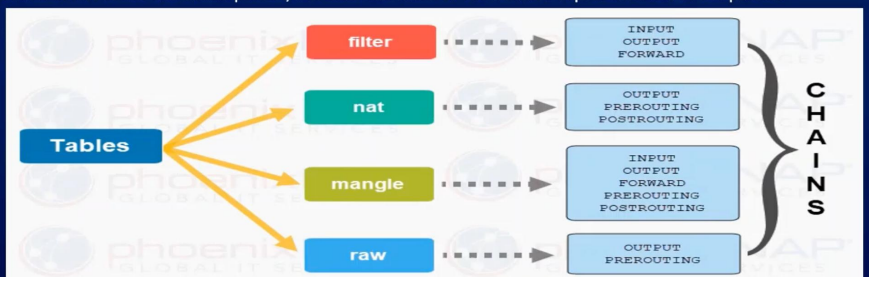
Quick intro to “iptables”

- The basic firewall software most commonly used in Linux is called iptables. The iptables firewall works by interacting with the packet filtering hooks in the Linux kernel’s networking stack. These kernel hooks are known as the netfilter framework.
- The iptables firewall uses **tables** to organize its rules. These tables classify rules according to the type of decisions they are used to make. For instance, if a rule deals with network address translation, it will be put into the nat table. If the rule is used to decide whether to allow the packet to continue to its destination, it will be added to the filter table.
- Within each iptables table, rules are further organized within separate **“chains”**. While tables are defined by the general aim of the rules they hold, the built-in chains represent the netfilter hooks which trigger them. ***Chains basically determine when rules will be evaluated***.
- A rule is a statement that tells the system what to do with a packet. Rules can block one type of packet, or forward another type of packet. The outcome, where a packet is sent, is called a **target**. A target is a decision of what to do with a packet. Typically, this is to accept it, drop it, or reject it.

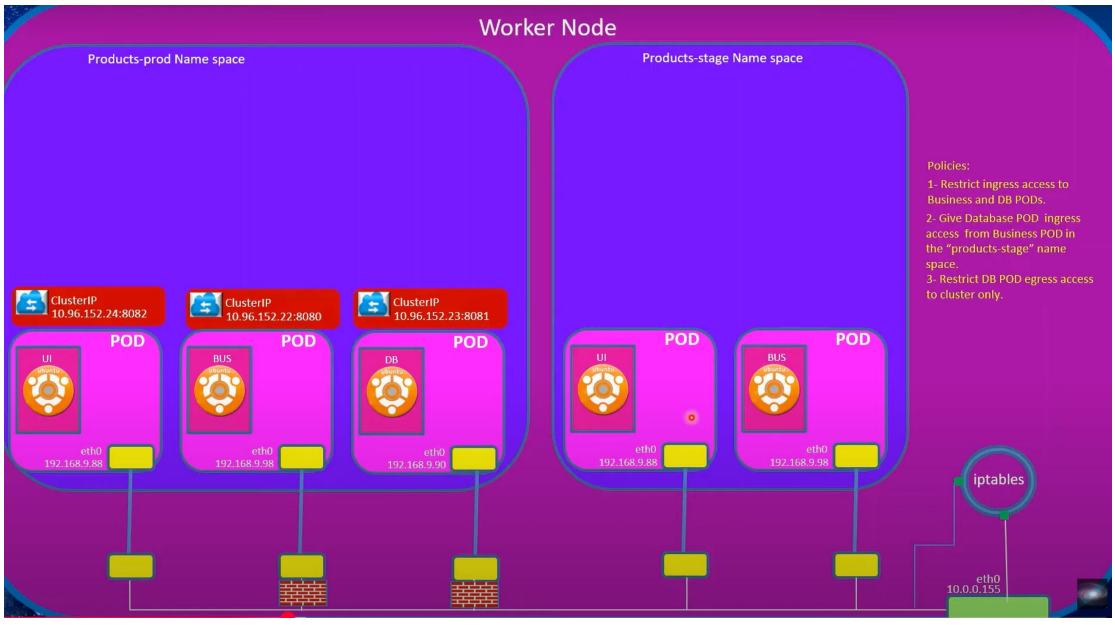
The Mangle Table

The mangle table is used to alter the IP headers of the packet in various ways. For instance, you can adjust the TTL (Time to Live) value of a packet, either lengthening or shortening the number of valid network hops the packet can sustain. Other IP headers can be altered in similar ways.

This table can also place an internal kernel “mark” on the packet for further processing in other tables and by other networking tools. This mark does not touch the actual packet, but adds the mark to the kernel’s representation of the packet.



Network policy



Network policy can select pods with matchLabels and set rules on ingress / Egress

restrict-access-to-business-tier-only.yaml

```yaml
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: restrict-access-to-business-tier-only
  namespace: products-prod
spec:
  podSelector:
    matchLabels:
      app: products-db
  ingress:
  - from:
    - podSelector:
        matchLabels:
          app: products-business
    ports:
      - protocol: TCP
        port: 8080
```

allow-products-prod-egress-traffic-to-cluster.yaml

```yaml
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-products-prod-egress-traffic-to-cluster
  namespace: products-prod
spec:
  podSelector:
    matchLabels: {}
  policyTypes:
  - Egress
  egress:
  - to:
    - ipBlock:
        cidr: 192.168.0.0/16
    - ipBlock:
        cidr: 10.0.0.0/16
```

```
apiVersion: projectcalico.org/v3
kind: NetworkPolicy
Metadata:
  name: cal-deny-ingress-from-ui
  namespace: products-prod
Spec:
  selector:  app == 'products-db'
  types:
  - Ingress
  - Egress
  ingress:
  - action: Log
    protocol: TCP
    source:
      selector: app == 'products-ui'
  - action: Deny
    protocol: TCP
    source:
      selector: app == 'products-ui'
```

Another Security enhancement is to use Service Account

```
apiVersion: v1
kind: ServiceAccount
metadata:
labels:
app.kubernetes.io/name: keda-operator
app.kubernetes.io/version: latest
app.kubernetes.io/part-of: keda-operator
name: keda-operator
namespace: keda
```

Roles : ClusterRoles and Name space Role

```
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
name: keda-operator
```

```
rules:
- apiGroups:
  - ""
  resources:
  - configmaps
  - configmaps/status
  verbs:
  - get
  - list
  - watch
- apiGroups:
  - ""
  resources:
  - events
  verbs:
  - '*'
- apiGroups:
  - ""
  resources:
  - external
  - pods
  - secrets
  - services
  verbs:
  - get
  - list
  - watch
- apiGroups:
  - ""
  resources:
  - limitranges
  verbs:
  - list
```

```
- watch
- apiGroups:
- ""
resources:
- serviceaccounts
verbs:
- list
- watch
- apiGroups:
- '*'
resources:
- '*'
verbs:
- get
- apiGroups:
- '*'
resources:
- '*/scale'
verbs:
- get
- list
- patch
- update
- watch
- apiGroups:
- admissionregistration.k8s.io
resources:
- validatingwebhookconfigurations
verbs:
- get
- list
- patch
- update
```

```
- watch
- apiGroups:
- apiregistration.k8s.io
resources:
- apiservices
verbs:
- get
- list
- patch
- update
- watch
- apiGroups:
- apps
resources:
- deployments
- statefulsets
verbs:
- list
- watch
- apiGroups:
- autoscaling
resources:
- horizontalpodautoscalers
verbs:
- '*'
- apiGroups:
- batch
resources:
- jobs
verbs:
- '*'
- apiGroups:
- eventing.keda.sh
```

```
resources:
- cloudeventsources
- cloudeventsources/status
verbs:
- '*'
- apiGroups:
- keda.sh
resources:
- clustertriggerauthentications
- clustertriggerauthentications/status
verbs:
- '*'
- apiGroups:
- keda.sh
resources:
- scaledjobs
- scaledjobs/finalizers
- scaledjobs/status
verbs:
- '*'
- apiGroups:
- keda.sh
resources:
- scaledobjects
- scaledobjects/finalizers
- scaledobjects/status
verbs:
- '*'
- apiGroups:
- keda.sh
resources:
- triggerauthentications
- triggerauthentications/status
```

```
verbs:
- '*'
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
name: keda-operator
namespace: keda
rules:
- apiGroups:
- ""
resources:
- secrets
verbs:
- create
- delete
- get
- list
- patch
- update
- watch
- apiGroups:
- coordination.k8s.io
resources:
- leases
verbs:
- '*'
```

Role Binding

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
name: keda-operator
```

```
roleRef:
apiGroup: rbac.authorization.k8s.io
kind: ClusterRole
name: keda-operator
subjects:
- kind: ServiceAccount
name: keda-operator
namespace: keda
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
name: keda-operator
namespace: keda
roleRef:
apiGroup: rbac.authorization.k8s.io
kind: Role
name: keda-operator
subjects:
- kind: ServiceAccount
name: keda-operator
namespace: keda
```

Deployment to control the replicate set of pods to create in the cluster

```
---
apiVersion: apps/v1
kind: Deployment
metadata:
name: keda-operator
namespace: keda
labels:
app: keda-operator
```

```
app.kubernetes.io/name: keda-operator
app.kubernetes.io/version: latest
app.kubernetes.io/component: operator
app.kubernetes.io/part-of: keda-operator
spec:
replicas: 1
selector:
matchLabels:
app: keda-operator
template:
metadata:
labels:
app: keda-operator
name: keda-operator
name: keda-operator
spec:
securityContext:
runAsNonRoot: true
serviceAccountName: keda-operator
containers:
- name: keda-operator
image: ghcr.io/kedacore/keda:latest
command:
- /keda
args:
- --leader-elect
- --zap-log-level=info
- --zap-encoder=console
- --zap-time-encoding=rfc3339
- --enable-cert-rotation=true
imagePullPolicy: Always
resources:
requests:
```

```
cpu: 100m
memory: 100Mi
limits:
cpu: 1000m
memory: 1000Mi
livenessProbe:
httpGet:
path: /healthz
port: 8081
initialDelaySeconds: 25
readinessProbe:
httpGet:
path: /readyz
port: 8081
initialDelaySeconds: 20
ports:
- containerPort: 8080
name: http
protocol: TCP
env:
- name: POD_NAMESPACE
valueFrom:
fieldRef:
fieldPath: metadata.namespace
- name: WATCH_NAMESPACE
value: ""
- name: KEDA_HTTP_DEFAULT_TIMEOUT
value: ""
securityContext:
runAsNonRoot: true
capabilities:
drop:
- ALL
```

```
allowPrivilegeEscalation: false
readOnlyRootFilesystem: true
seccompProfile:
type: RuntimeDefault
volumeMounts:
- mountPath: /certs
name: certificates
readOnly: true
terminationGracePeriodSeconds: 10
nodeSelector:
kubernetes.io/os: linux
volumes:
- name: certificates
secret:
defaultMode: 420
secretName: kedaorg-certs
optional: true
```

Service is to create the network layer which has the endpoints to the pods created by the deployment

```
---
apiVersion: v1
kind: Service
metadata:
labels:
app.kubernetes.io/name: keda-operator
app.kubernetes.io/version: latest
app.kubernetes.io/part-of: keda-operator
name: keda-operator
namespace: keda
spec:
ports:
```

```
- name: metricsservice
port: 9666
targetPort: 9666
- name: metrics
port: 8080
targetPort: 8080
selector:
app: keda-operator
```

References

1. Kubernetes Authors. (2023). *Network Policies*. https://kubernetes.io/docs/concepts/services-networking/network-policies/

2. Project Calico. (2023). *BGP Configuration Guide*. https://docs.projectcalico.org/networking/bgp

3. Li, W., et al. (2020). "Microsegmentation in Cloud Networks." *IEEE Transactions on Cloud Computing*, 8(2), 456-470. https://doi.org/10.1109/TCC.2020.2988001

4. NSA/CISA. (2021). *Kubernetes Hardening Guidance*. https://media.defense.gov/2021/Aug/03/2002820425/-1/-1/1/CTR_KUBERNETES_HARDENING_GUIDANCE.PDF

5. CNI Maintainers. (2023). *Container Network Interface Specification*. https://github.com/containernetworking/cni