ESCOLA SUPERIOR DE REDES

Orquestração de Containers

Redes no Kubernetes

Tópicos abordados

- Modelo de redes no Kubernetes
- Services e Ingress
- CNI
- Calico
- kube-proxy
- CoreDNS

O k8s usa uma abordagem diferente para gerência de redes: **terceiriza através de plugins**

Coordenar a entrega de IPs e portas é extremamente complexo em ambientes de larga escala

CNI - Container Network Interface

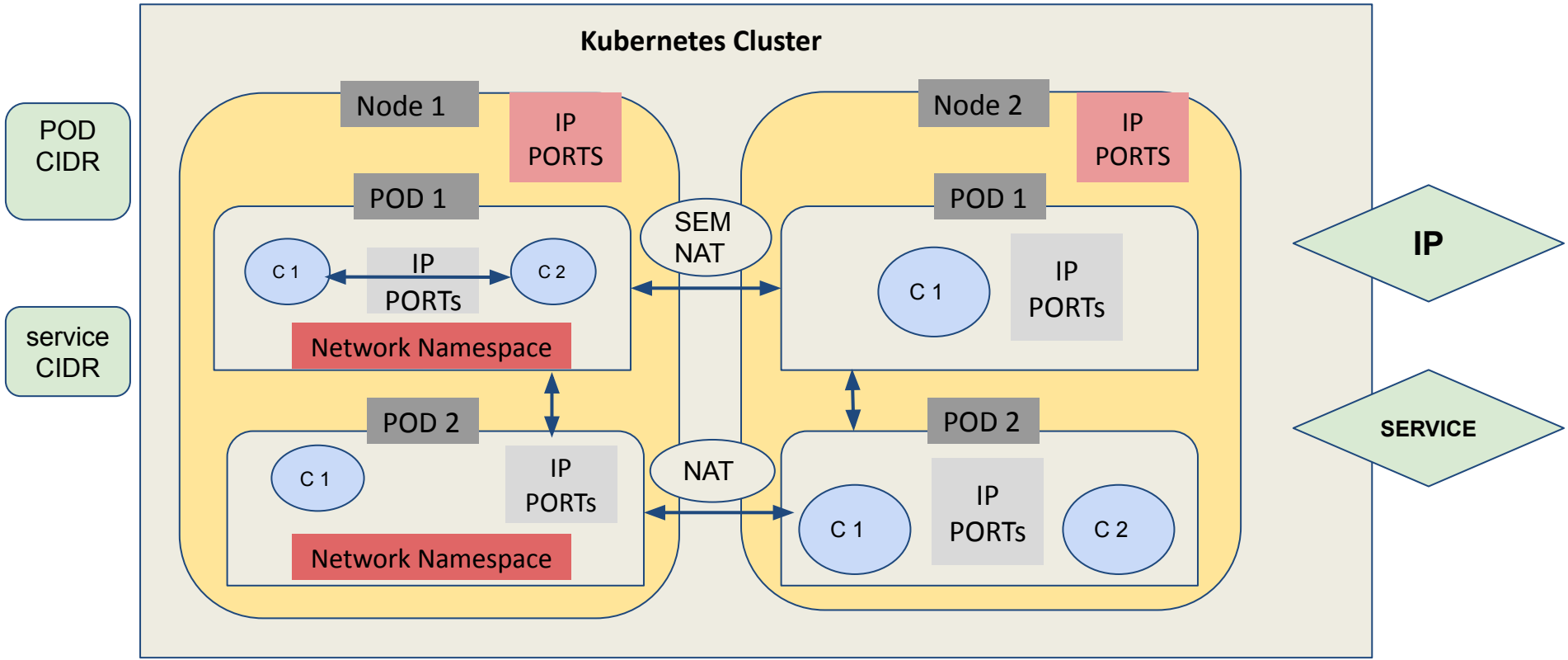
Redes no k8s visam oferecer a menor resistência possível para portar aplicações executadas em VMs. Para esse fim:

Cada POD possui um endereço IP específico (Network Namespace)

PODs podem se comunicar entre si através de IP em todos os NODES sem utlizar NAT

Agentes em um NODE (kubelet ou um serviço local) podem se comunicar com todos os PODs daquele NODE

Modelo de Redes no Kubernetes



Como descobrir POD CIDR e service CIDR ?

kubectl cluster-info dump | egrep 'service-cluster-ip-range|cluster-cidr'

kubectl -n kube-system describe configmaps kubeadm-config | egrep -i 'podsubnet|servicesubnet'

/etc/kubernetes/manifests/kube-apiserver.yaml -> --service-cluster-ip-range

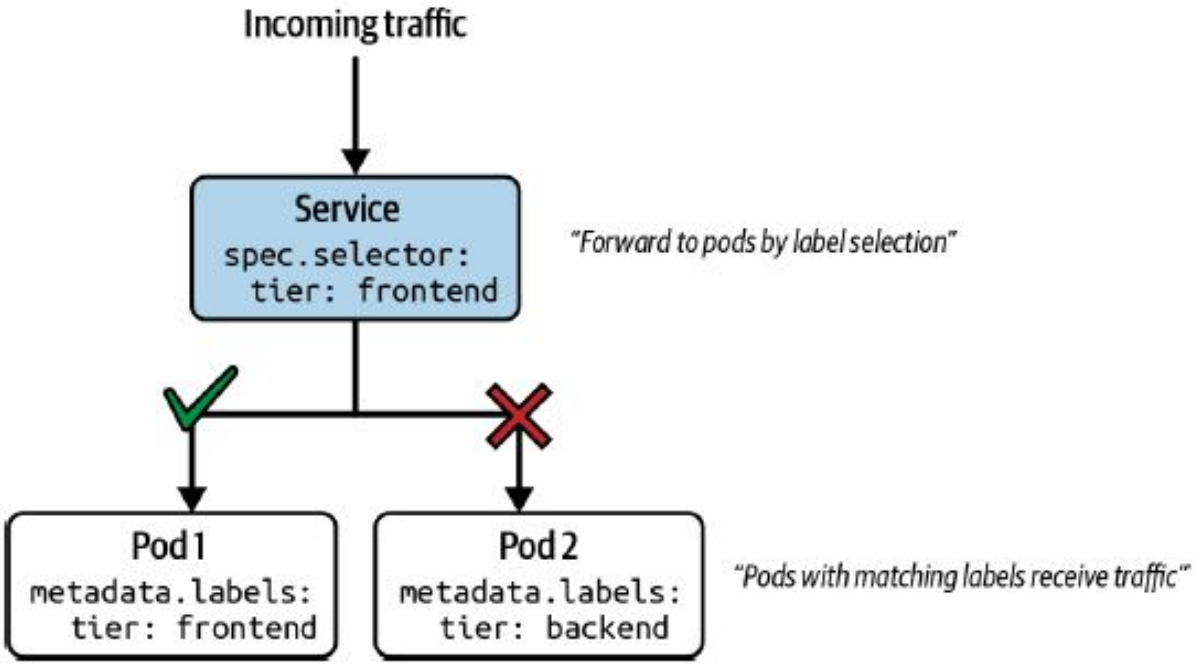
via plugin CNI

Quando definir POD CIDR e service CIDR ?

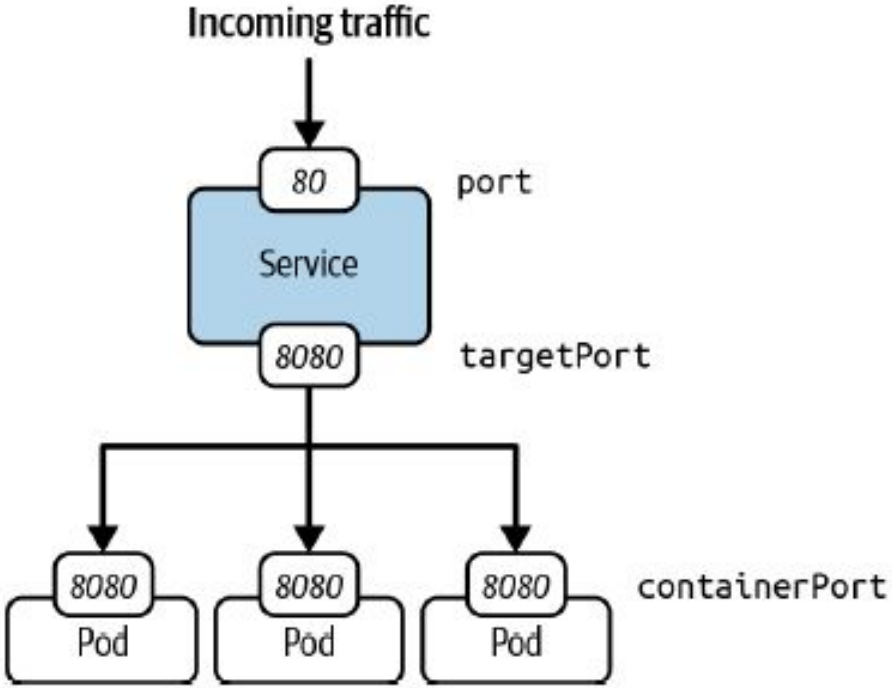
Setup do cluster:

Ex: com kubeadm:
kubeadm init --pod-network-cidr=<net/mask> --service-cidr=<net/mask>

- IP não é estável dentro do cluster - PODs são efêmeros
- Uma aplicação pode ter várias réplicas
- Necessidade de DNS interno -> CoreDNS



Service - Port Mapping



Service

kubectl describe service web

```
Name:              web
Namespace:         teste
Labels:            <none>
Annotations:       <none>
Selector:          app=web
Type:              ClusterIP
IP Family Policy:  SingleStack
IP Families:       IPv4
IP:                10.102.125.48
IPs:               10.102.125.48
Port:              <unset>  8080/TCP
TargetPort:        80/TCP
Endpoints:         10.32.0.3:80,10.44.0.2:80
Session Affinity:  None
Events:            <none>
```

IP VRITUAL
KUBE-PROXY

IP do POD

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: web
  name: web
  namespace: teste
spec:
  replicas: 1
  selector:
    matchLabels:
      app: web
  template:
    metadata:
      labels:
        app: web
    spec:
      containers:
      - image: nginx
        name: web
        ports:
        - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
  name: web
  namespace: teste
spec:
  type: ClusterIP
  ports:
  - port: 8080
    protocol: TCP
    targetPort: 80
  selector:
    app: web
```

Label Selector

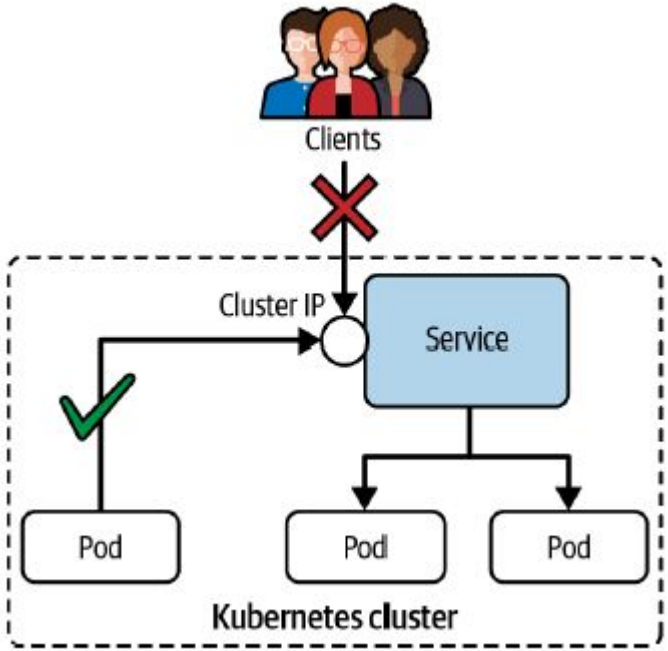
EndPoint Port

Service Port

Tipos de Services

ClusterIP

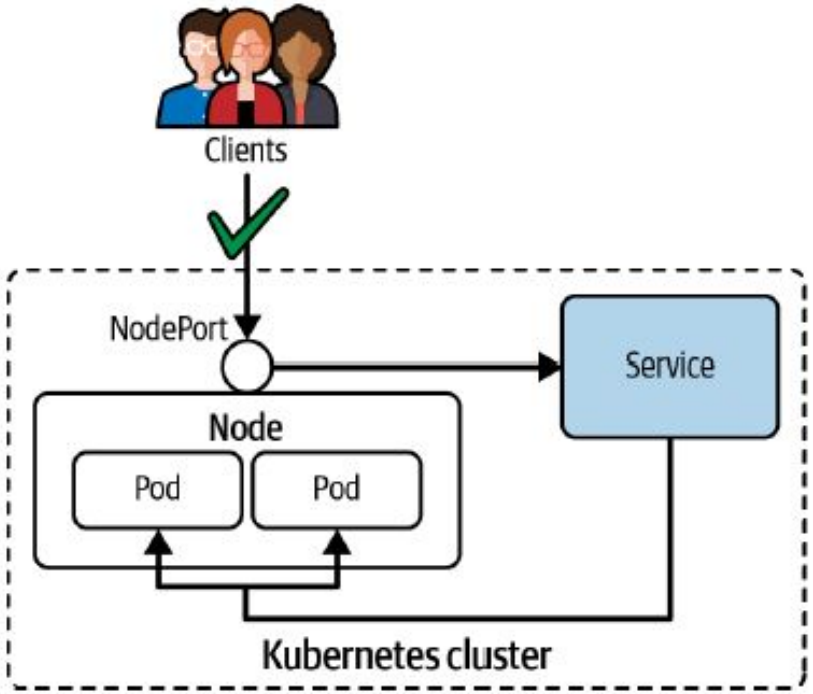
- Tipo padrão
- Somente acesso interno ao cluster



Tipos de Services

NodePort

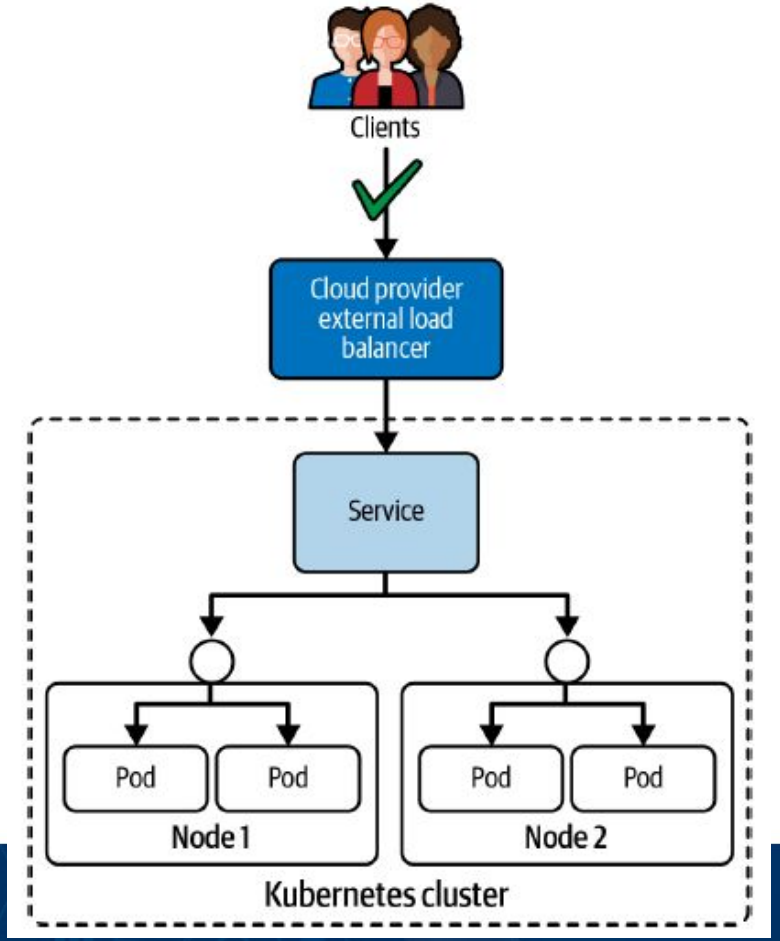
- Expõe uma porta em cada um dos nodes para acesso externo ao cluster
- Intervalor de portas: 30000 - 32767



Tipos de Services

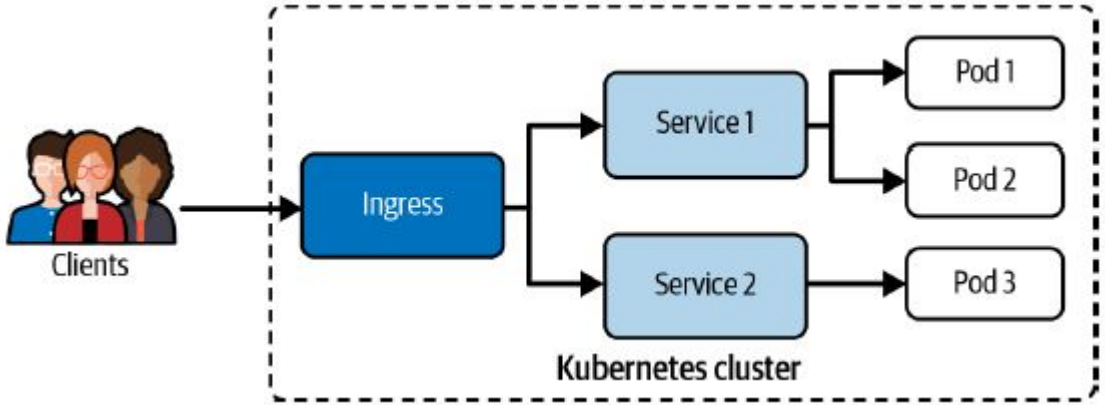
LoadBalancer

- Expõe um IP externo: endereço público provido pelo provedor de nuvem
- Geralmente utilizado com ingress controller



Ingress

- Expõe rotas de aplicações HTTP/HTTPS para fora do cluster



Objeto de API que gerencia acesso externo para acesso a serviços dentro do *cluster*

Pode prover funcionalidades como *load balancing*, SSL *termination* e *name-based virtualhosting*

Expõe rotas HTTP e HTTPS de fora do *cluster* para serviços

Roteamento é controlado por regras definidas dentro do recurso *Ingress*

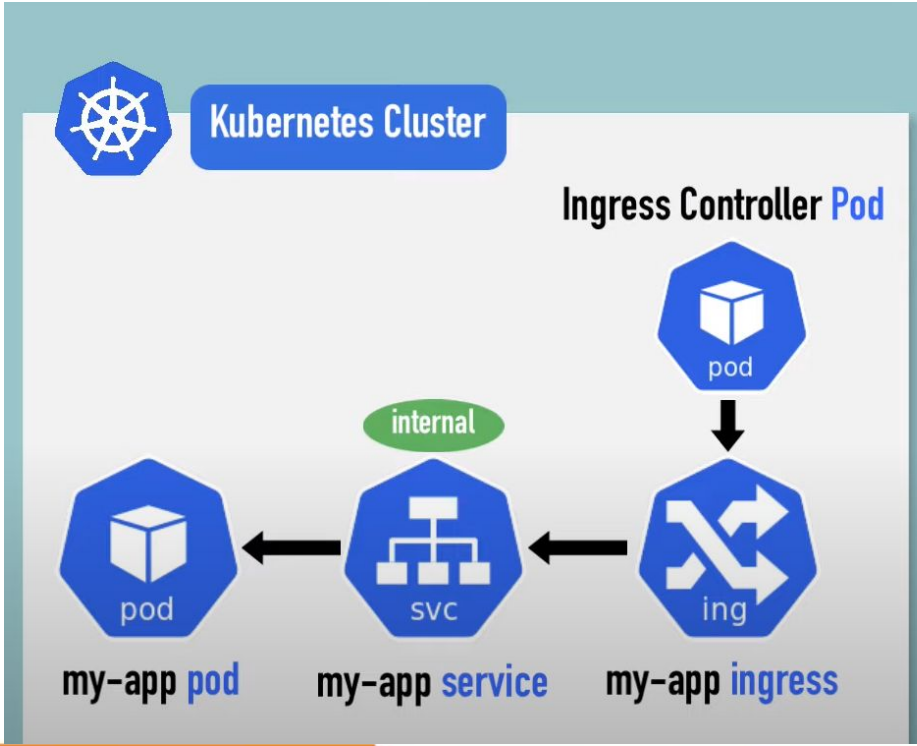
Ingress


cluster
client
Ingress-managed
load balancer
Ingress
routing rule
Service
Pod
Pod

Ingress Controller

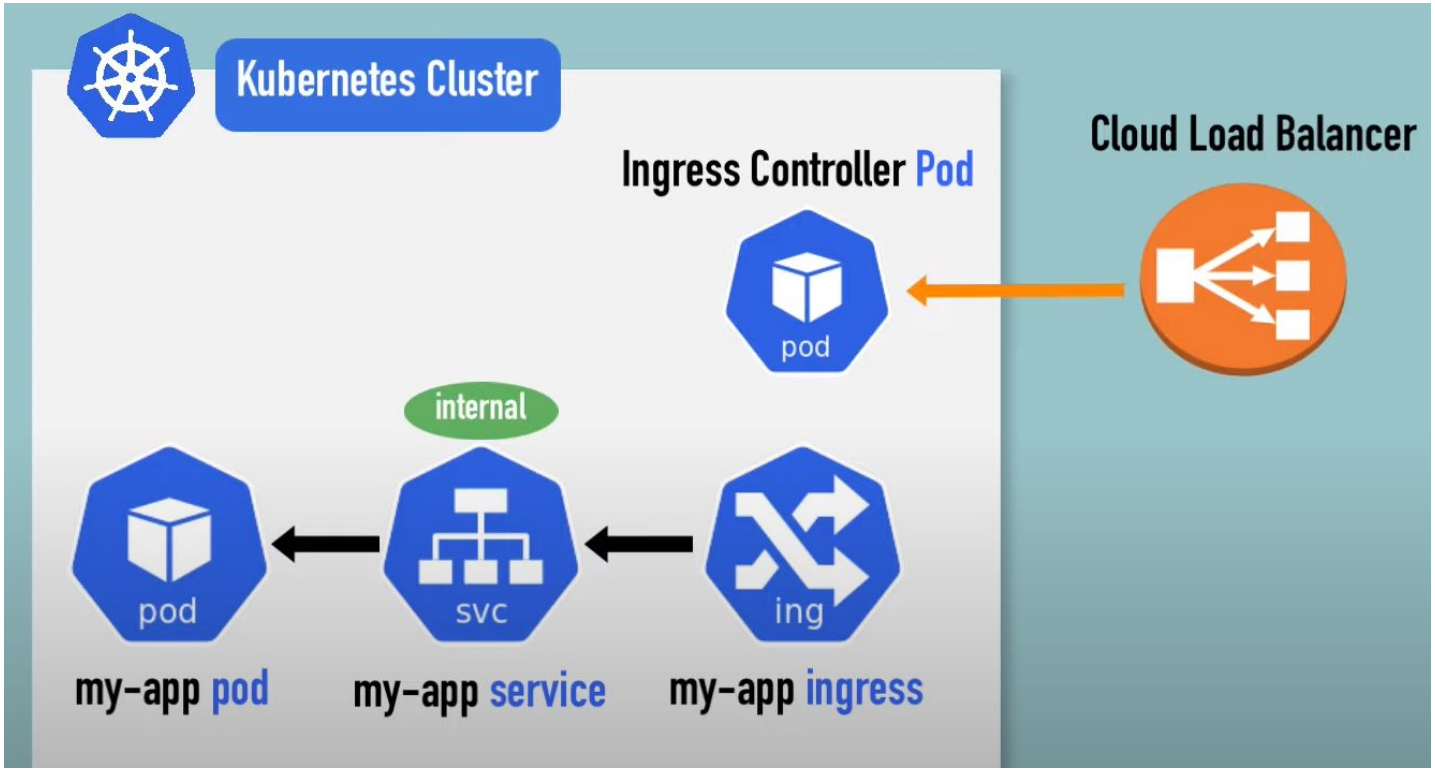
Para que o *Ingress* funcione, é primeiro necessário ter um *ingress controller* operacional no *cluster*

Ingress Controller



- evaluates all the rules
- manages redirections
- entrypoint to cluster
- many third-party implementations
- K8s Nginx Ingress Controller

Ingress Controller



Ingress Controller

Azure AKS	Contour	Istio
Ambassador	EnRoute	Kong
Apace APISIX	F5 BIG-IP	**NGINX**
Avi Operator	Gloo	Skipper
Citrix	HAProxy	Traefik

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

Criando o *Ingress* via arquivo YAML

kubectl get ingressclasses

Antes de criar o manifesto do ingress, é necessário primeiro instalar o:
ingressClass

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: minimal-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  ingressClassName: nginx-example
  rules:
  - http:
      paths:
      - path: /testpath
        pathType: Prefix
        backend:
          service:
            name: test
            port:
              number: 80
```

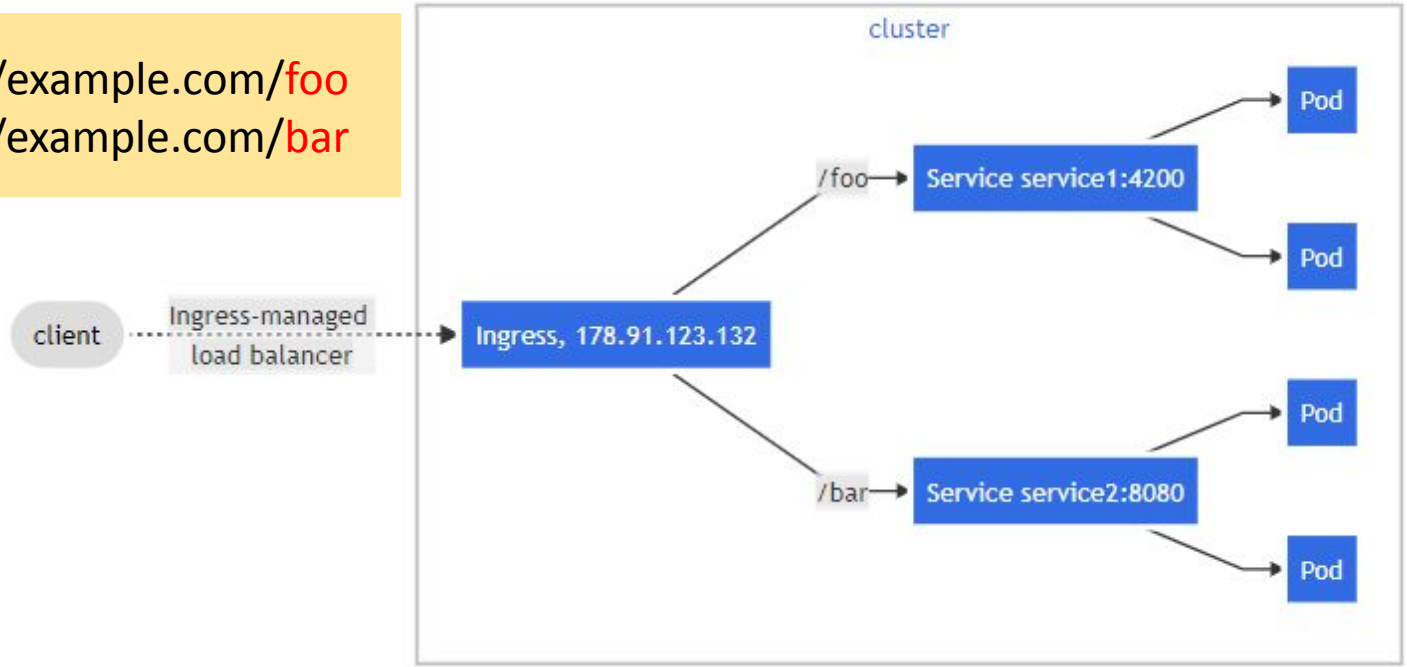
domínio do host

kubernetes service

geralmente 80 ou 443

Topologias para *Ingress: simple fanout*

https://example.com/foo
https://example.com/bar



Topologias para *Ingress: simple fanout*

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: simple-fanout-example
spec:
  rules:
  - host: foo.bar.com
    http:
      paths:
      - path: /foo
        pathType: Prefix
        backend:
          service:
            name: service1
            port:
              number: 4200
```

```yaml
      - path: /bar
        pathType: Prefix
        backend:
          service:
            name: service2
            port:
              number: 8080
```

Topologias para *Ingress: simple fanout*

6.2) Ingress via HTTP

Uma vez criado o *Ingress Controller* podemos, agora sim, criar objetos Ingress e distribuir o tráfego para diferentes aplicações. Vamos fazer isso.

a. Claro, precisamos de aplicações primeiro. Siga os passos abaixo:

- Crie o namespace `color`.
- Dentro dele, crie o deployment `blue` contendo apenas uma réplica usando a imagem `fbscarel/myapp-color:blue`. Crie também um serviço que exponha esse deployment dentro do contexto do *cluster* na porta 80.
- De igual forma, crie o deployment `red` usando a imagem `fbscarel/myapp-color:red` com as mesmas características do item anterior (incluindo o serviço).

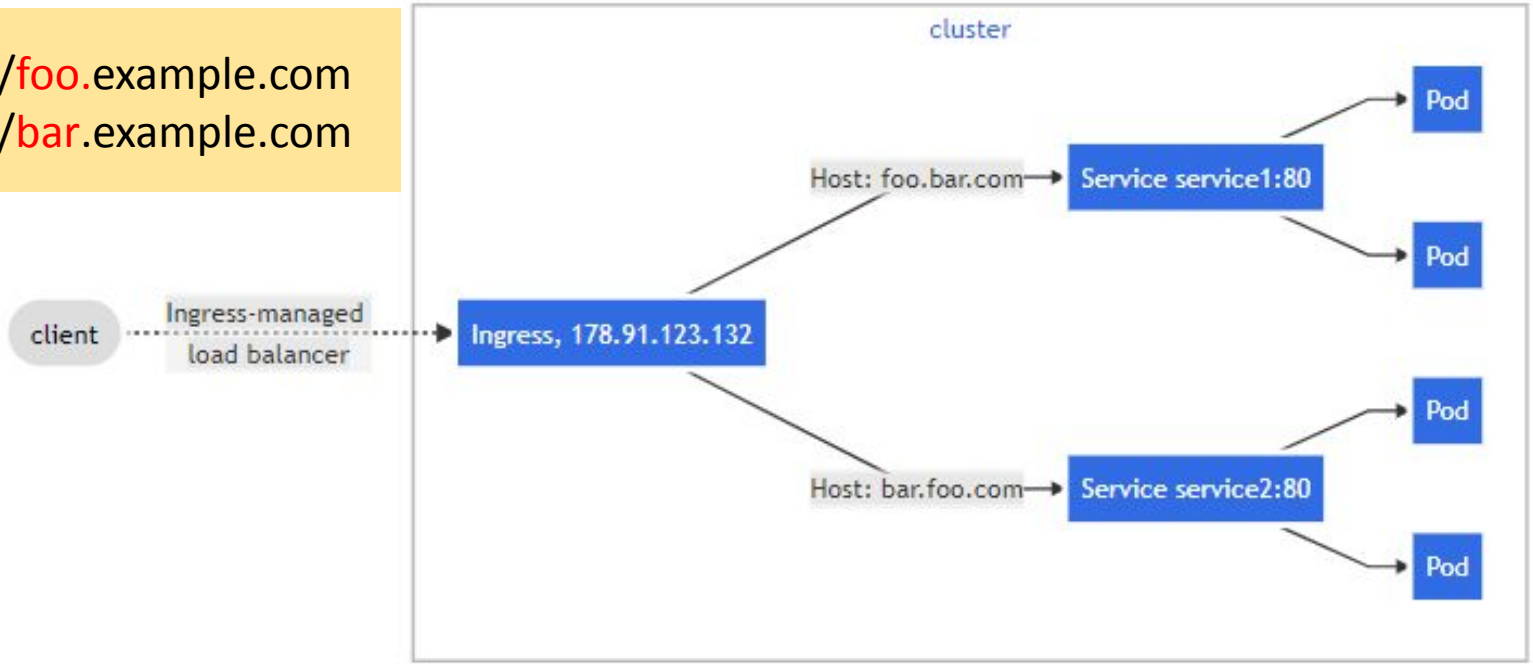
Verifique o funcionamento de sua configuração antes de prosseguir.

▸ Visualizar resposta

b. Agora, é hora de criar o Ingress. Imagine que queremos acessar a URL `color.contorq.com`, e dentro desse website as páginas `/blue` e `/red` irão apontar para cada um dos deployments realizados no passo anterior.

Topologias para *Ingress: name-based virtualhosting*

https://foo.example.com
https://bar.example.com



Topologias para *Ingress: name-based virtualhosting*

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: name-virtual-host-ingress
spec:
  rules:
  - host: foo.bar.com
    http:
      paths:
      - pathType: Prefix
        path: "/"
        backend:
          service:
            name: service1
            port:
              number: 80
```

```yaml
  - host: bar.foo.com
    http:
      paths:
      - pathType: Prefix
        path: "/"
        backend:
          service:
            name: service2
            port:
              number: 80
```

Topologias para *Ingress: name-based virtualhosting*

6.3) Ingress via HTTPS

a. Vamos agora incrementar a configuração realizada durante a atividade anterior. Para começar, crie um certificado auto-assinado usando o comando `openssl req`. Na linha *Subject*, garanta que os atributos `CN` e `O` correspondem a *wildcard* `*.color.contorq.com`.

▶ Visualizar resposta

b. Crie o secret `color-tls` utilizando a chave privada e certificado criado no passo anterior. Utilize o tipo `tls`.

A seguir, verifique o funcionamento de sua configuração.

▶ Visualizar resposta

c. Remova o Ingress `ingress-color-name` e, em seu lugar, crie o Ingress `ingress-color-tls` que sirva os recursos através das URLs `blue.color.contorq.com` e `red.color.contorq.com`.

Utilizando HTTPS no *Ingress*

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: testsecret-tls
  namespace: default
data:
  tls.crt: base64 encoded cert
  tls.key: base64 encoded key
type: kubernetes.io/tls
```

Especial atenção à correta configuração do CN do certificado!

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: tls-example-ingress
spec:
  tls:
  - hosts:
      - https-example.foo.com
    secretName: testsecret-tls
  rules:
  - host: https-example.foo.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: service1
            port:
              number: 80
```

Utilizando HTTPS no *Ingress*

cert-manager

Automated Kubernetes
X.509 Certificates

https://cert-manager.io/v1.1-docs/installation/kubernetes
https://cert-manager.io/v1.1-docs/tutorials/acme/ingress

Exemplo prático

Atividade sessão 8

6.1) Ingress Controller

6.2) Ingress HTTP

6.3) Ingress HTTPS

Em lugar de solucionar o problema de redes, o k8s o terceiriza.

Devem-se usar *plugins* CNI para esse fim.

https://kubernetes.io/pt-br/docs/concepts/extend-kubernetes/compute-storage-net/network-plugins

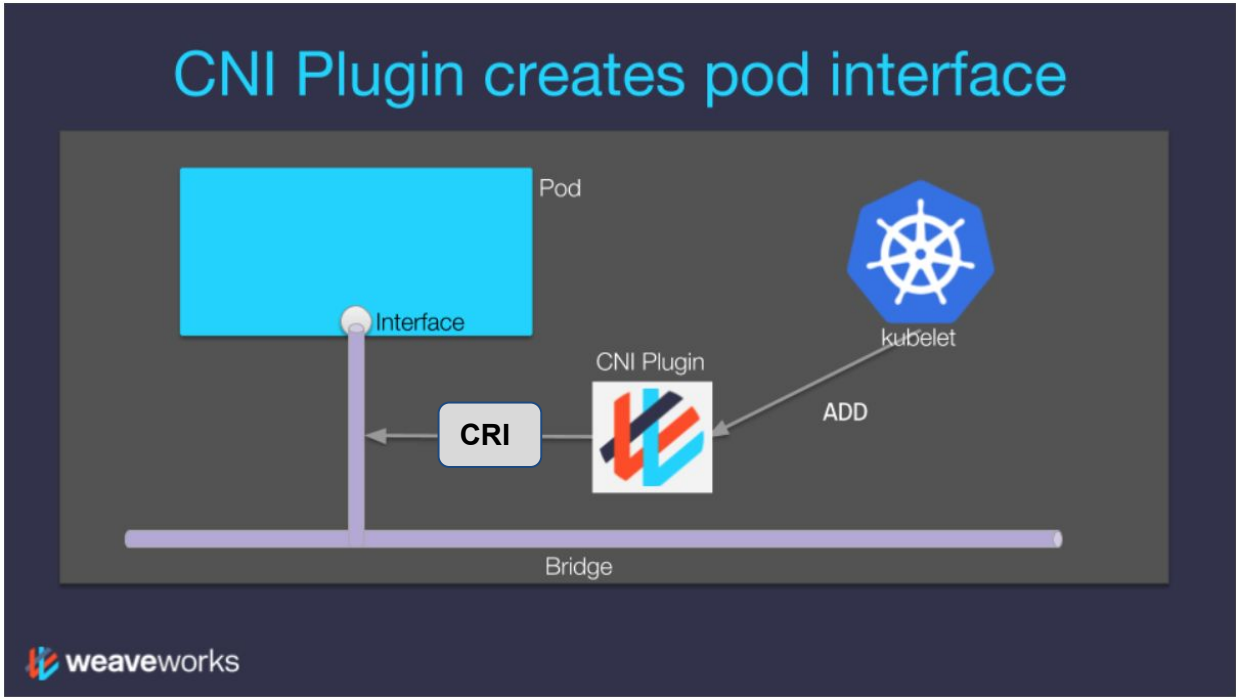
Projeto da CNCF contendo especificações e bibliotecas para escrita de *plugins* de rede para o k8s e similares

Por ser uma especificação simples, permite uma grande gama de soluções

A ideia básica é propiciar uma interface tão dinâmica quanto a velocidade de mudanças da tecnologia

CNI *plugins*

Cisco ACI	Big Cloud	Contiv
Antrea	**Calico**	Contrail
Apstra AOS	**Cilium**	DANM
AWS VPC	cni-ipvlan-vpc	**Flannel**
Azure CNI	**Weave Net**	GCE CNI


CNI Plugin creates pod interface
Pod
Interface
kubelet
CNI Plugin
CRI
ADD
Bridge
weaveworks

Calico

Control plane que suporta múltiplos dataplanes de operação

Solução de rede L2/L3 que conecta containers, clusters k8s, VMs e hosts físicos

Oferece suporte a *network policies* e auto-descoberta de *hosts*

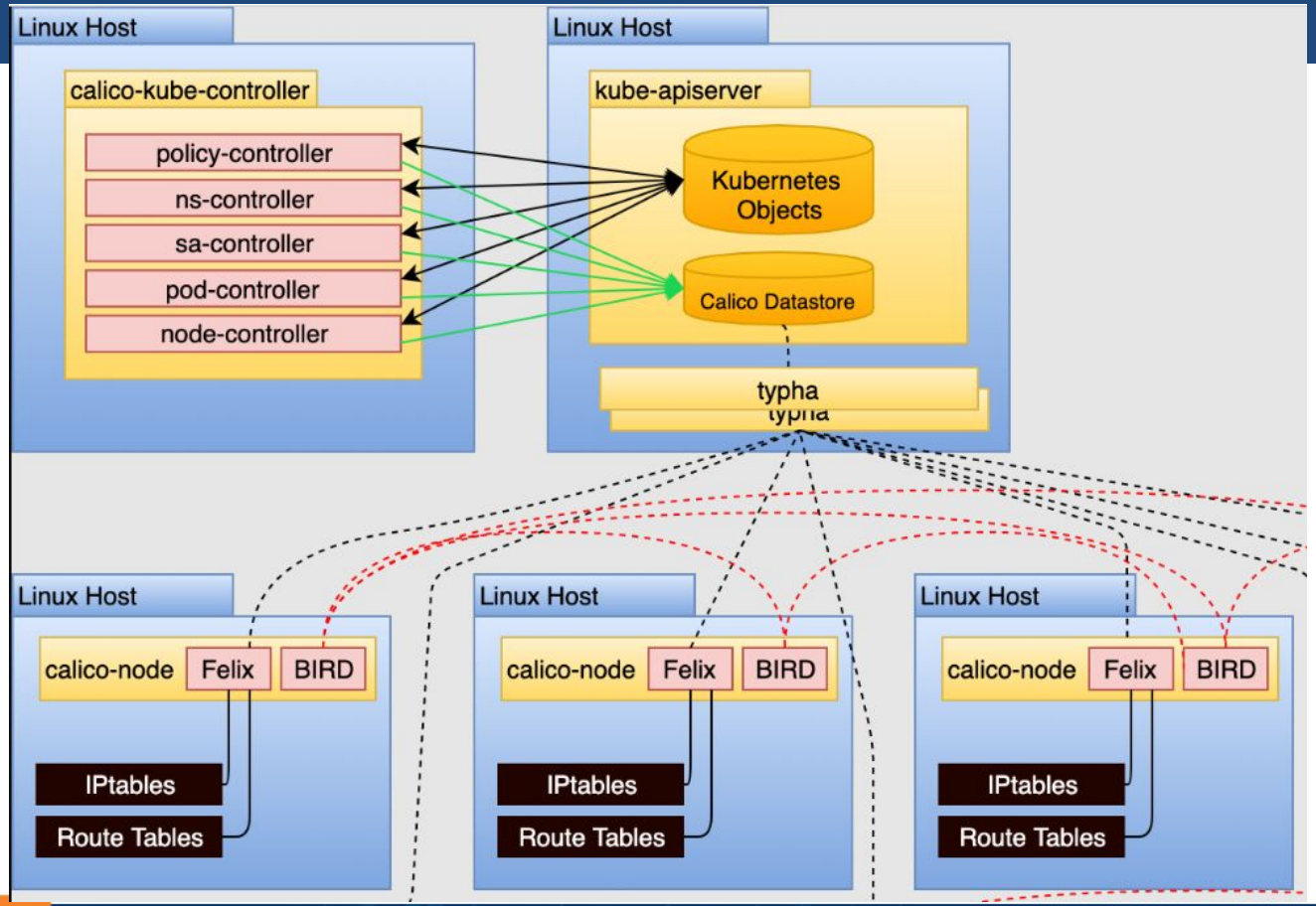
Fácil configuração, instalado via um único comando *kubectl apply*

https://docs.tigera.io/calico/latest/about

Instalação do Calico

https://docs.tigera.io/calico/latest/getting-started/kubernetes/self-managed-onprem/onpremises

Arquitetura



https://tanzu.vmware.com/developer/guides/container-networking-calico-refarch

Visualizando informações sobre o Calico

Componentes dentro do cluster

Configuração IPAM

Faixa de endereços alocada pelo Calico para *pods*

https://docs.tigera.io/calico/latest/reference/configure-cni-plugins

Visualizando informações sobre o Calico

```
root@s2-master-1:~# k get all -A | grep calico
kube-system   pod/calico-kube-controllers-674fff74c8-bnj5z   1/1     Running
kube-system   pod/calico-node-dfspv                          1/1     Running
kube-system   pod/calico-node-zskq2                          1/1     Running
kube-system   daemonset.apps/calico-node   2         2         2       2
kube-system   deployment.apps/calico-kube-controllers   1/1     1            1
kube-system   replicaset.apps/calico-kube-controllers-674fff74c8   1         1
```

Visualizando informações sobre o Calico

deployment/calico-kube-controllers:

1. policy controller: observa as políticas de rede e programa as políticas do calico.
2. namespace controller: observa namespaces e programa perfis Calico.
3. service account controller: observa contas de serviço e programa perfis Calico.
4. workload endpoint controller: observa alterações nos rótulos dos pods e atualiza os endpoints de carga de trabalho do Calico.
5. node controller: observa a remoção dos nós do Kubernetes e remove os dados correspondentes do Calico.

https://docs.tigera.io/archive/v3.10/reference/kube-controllers/configuration

Visualizando informações sobre o Calico

daemonset/calico-node:

1. Felix: daemon Calico que é executado em todos os nós e fornece terminais.
2. BIRD: daemon BGP que distribui informações de roteamento para outros nós.
3. confd: daemon que observa o armazenamento de dados do Calico em busca de alterações de configuração e atualiza os arquivos de configuração do BIRD.

https://docs.tigera.io/calico/latest/reference/configure-calico-node

Visualizando informações sobre o Calico

CRDs do Calico

```
root@s2-master-1:~# k api-resources | grep calico
bgpconfigurations                 crd.projectcalico.org/v1   false   BGPConfiguration
bgppeers                          crd.projectcalico.org/v1   false   BGPPeer
blockaffinities                   crd.projectcalico.org/v1   false   BlockAffinity
caliconodestatuses                crd.projectcalico.org/v1   false   CalicoNodeStatus
clusterinformations               crd.projectcalico.org/v1   false   ClusterInformation
felixconfigurations               crd.projectcalico.org/v1   false   FelixConfiguration
globalnetworkpolicies             crd.projectcalico.org/v1   false   GlobalNetworkPolicy
globalnetworksets                 crd.projectcalico.org/v1   false   GlobalNetworkSet
hostendpoints                     crd.projectcalico.org/v1   false   HostEndpoint
ipamblocks                        crd.projectcalico.org/v1   false   IPAMBlock
ipamconfigs                       crd.projectcalico.org/v1   false   IPAMConfig
ipamhandles                       crd.projectcalico.org/v1   false   IPAMHandle
ippools                           crd.projectcalico.org/v1   false   IPPool
ipreservations                    crd.projectcalico.org/v1   false   IPReservation
kubecontrollersconfigurations     crd.projectcalico.org/v1   false   KubeControllersConfiguration
networkpolicies                   crd.projectcalico.org/v1   true    NetworkPolicy
networksets                       crd.projectcalico.org/v1   true    NetworkSet
```

POD CIDR

Visualizando informações sobre o Calico

```
root@s2-master-1:~# k get ippools
NAME                  AGE
default-ipv4-ippool   33d
root@s2-master-1:~# k describe ippools default-ipv4-ippool
Name:         default-ipv4-ippool
Namespace:
Labels:       <none>
Annotations:  projectcalico.org/metadata: {"uid":"4b0216b6-7f33-4f
API Version:  crd.projectcalico.org/v1
Kind:         IPPool
Metadata:
  Creation Timestamp:  2023-07-12T12:59:42Z
  Generation:          1
  Resource Version:    867
  UID:                 d0d64f3d-f2f1-4f30-96a1-7c5dcd390af2
Spec:
  Allowed Uses:
    Workload
    Tunnel
  Block Size:     26
  Cidr:           10.32.0.0/12
  Ipip Mode:      Always
  Nat Outgoing:   true
  Node Selector:  all()
  Vxlan Mode:     Never
Events:           <none>
```

Visualizando informações sobre o Calico

```
root@s2-master-1:~# k get blockaffinities
NAME                              AGE
s2-master-1-10-45-193-192-26      33d
s2-node-1-10-41-181-128-26        33d
root@s2-master-1:~# k describe blockaffinities s2-node-1-10-41-181-128-26
Name:         s2-node-1-10-41-181-128-26
Namespace:
Labels:       <none>
Annotations:  projectcalico.org/metadata: {"creationTimestamp":null}
API Version:  crd.projectcalico.org/v1
Kind:         BlockAffinity
Metadata:
  Creation Timestamp:  2023-07-12T13:01:02Z
  Generation:          2
  Resource Version:    1070
  UID:                 483b8582-ba85-4fc1-8574-58536f05420a
Spec:
  Cidr:     10.41.181.128/26
  Deleted:  false
  Node:     s2-node-1
  State:    confirmed
Events:     <none>
```

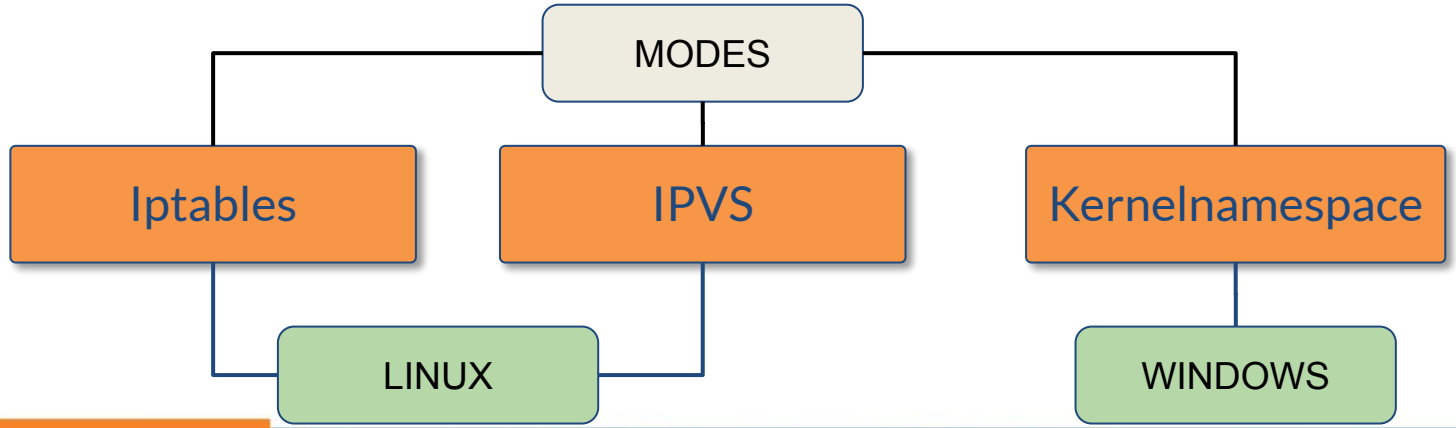
Visualizando informações sobre o Calico

```
root@s2-master-1:~# k get ipamblocks
NAME                 AGE
10-41-181-128-26     33d
10-45-193-192-26     33d
root@s2-master-1:~# k describe ipamblocks 10-41-181-128-26
Name:         10-41-181-128-26
Namespace:
Labels:       <none>
Annotations:  projectcalico.org/metadata: {"creationTimestamp":null}
API Version:  crd.projectcalico.org/v1
Kind:         IPAMBlock
Metadata:
  Creation Timestamp:  2023-07-12T13:01:02Z
  Generation:          551
  Resource Version:    498088
  UID:                 5b7c502a-2d78-471b-bea6-4616933905a6
Spec:
  Affinity:  host:s2-node-1
  Allocations:
    0
    <nil>
```

Kube-proxy

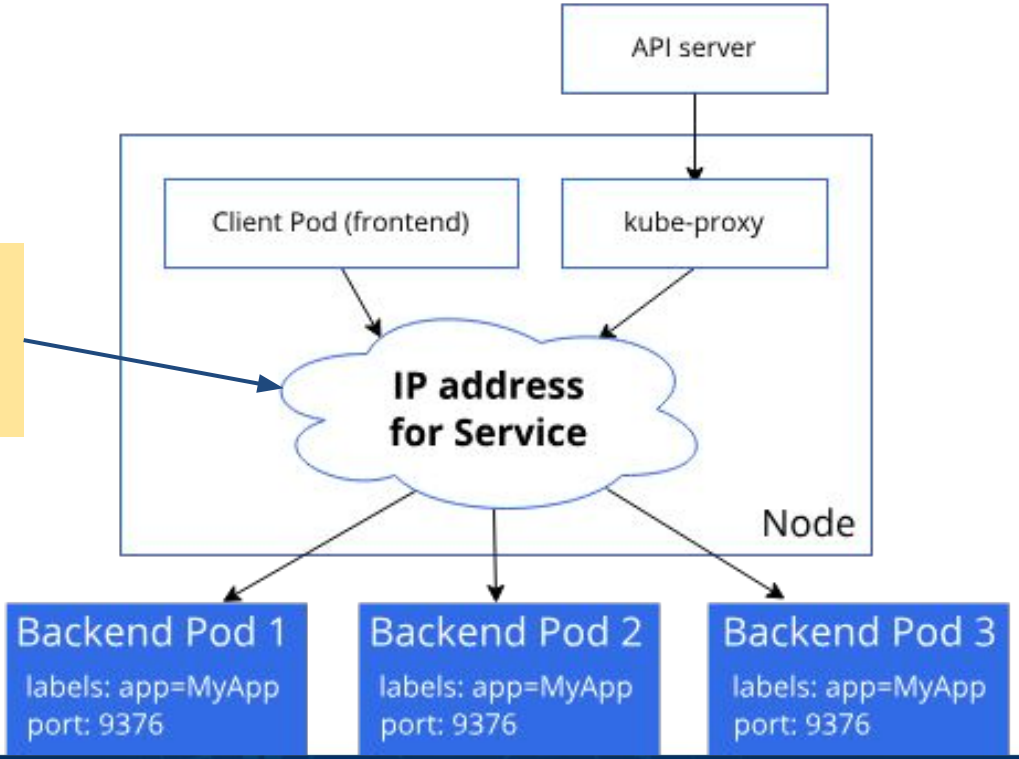
O *kube-proxy* é um *proxy* de rede que opera em cada um dos nodes do *cluster*

IP virtual dos *services*
Load Balancer
NAT



Funcionamento do *kube-proxy* Mode Iptables

IPTABLES
-> TABELA NAT
-> TARGET DNAT



Funcionamento do *kube-proxy*
Mode Iptables

```
root@s2-master-1:~# k apply -f manifests/proxy-clusterip.yaml
service/proxy-clusterip created
pod/proxy-clusterip created
root@s2-master-1:~# k get pod
NAME              READY   STATUS    RESTARTS   AGE
proxy-clusterip   1/1     Running   0          11s
root@s2-master-1:~# k get svc
NAME              TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE
proxy-clusterip   ClusterIP   10.103.96.239   <none>        8080/TCP   15s
```

Funcionamento do *kube-proxy* Mode Iptables

```
root@s2-master-1:~# k run curl -it --image=curlimages/curl -- sh
If you don't see a command prompt, try pressing enter.
~ $ curl -s proxy-clusterip:8080
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

Funcionamento do *kube-proxy*
Mode Iptables

```
root@s2-master-1:~# iptables -t nat -nv -L | grep ':8080' | grep -v MARK
    0     0 KUBE-SVC-D2KB74CYAGKPUNAI  tcp  --  *      *       0.0.0.0/0            10.103.96.239        /* proxy/proxy-clusterip cluster IP */ tcp dpt:8080
root@s2-master-1:~# iptables -t nat -nv -L KUBE-SVC-D2KB74CYAGKPUNAI | grep -v MARK
Chain KUBE-SVC-D2KB74CYAGKPUNAI (1 references)
 pkts bytes target     prot opt in     out     source               destination
    0     0 KUBE-SEP-GQ4GIFESINNSO3KL  all  --  *      *       0.0.0.0/0            0.0.0.0/0            /* proxy/proxy-clusterip -> 10.41.181.161:80 */
root@s2-master-1:~# iptables -t nat -nv -L KUBE-SEP-GQ4GIFESINNSO3KL | grep -v MARK
Chain KUBE-SEP-GQ4GIFESINNSO3KL (1 references)
 pkts bytes target     prot opt in     out     source               destination
    0     0 DNAT       tcp  --  *      *       0.0.0.0/0            0.0.0.0/0            /* proxy/proxy-clusterip */ tcp to:10.41.181.161:80
```

Arquivo de configuração

```
root@s2-master-1:~# k -n kube-system describe ds kube-proxy
Name:           kube-proxy
Selector:       k8s-app=kube-proxy
```

```
    /usr/local/bin/kube-proxy
    --config=/var/lib/kube-proxy/config.conf
    --hostname-override=$(NODE_NAME)
```

```
Volumes:
 kube-proxy:
  Type:      ConfigMap (a volume populated by a ConfigMap)
  Name:      kube-proxy
  Optional:  false
```

Arquivo de configuração

```
root@s2-master-1:~# k -n kube-system get configmaps kube-proxy -o yaml
apiVersion: v1
data:
  config.conf: |-
    apiVersion: kubeproxy.config.k8s.io/v1alpha1
    bindAddress: 0.0.0.0
    bindAddressHardFail: false
    clientConnection:
      acceptContentTypes: ""
      burst: 0
      contentType: ""
      kubeconfig: /var/lib/kube-proxy/kubeconfig.conf
      qps: 0
    clusterCIDR: 10.32.0.0/12
    configSyncPeriod: 0s
```

Esse parâmetro vem da onde?

Exemplo prático

Como o *kube-proxy* implementa acesso a serviços publicados no *cluster*?

Atividade sessão 8:
Número 4, letra e

Servidor DNS extensível utilizado como *resolver* para o *cluster* Kubernetes

Praticamente todas as funcionalidades são externalizadas para *plugins*

A descoberta de serviços no k8s é implementada via ***plugin* kubernetes**

Arquivo de configuração

Domínio raiz do cluster

```
root@s2-master-1:~# k -n kube-system get configmap coredns -o yaml
apiVersion: v1
data:
  Corefile: |
    .:53 {
        errors
        health {
           lameduck 5s
        }
        ready
        kubernetes cluster.local in-addr.arpa ip6.arpa {
           pods insecure
           fallthrough in-addr.arpa ip6.arpa
           ttl 30
        }
        prometheus :9153
        forward . /etc/resolv.conf {
           max_concurrent 1000
        }
        cache 30
        loop
        reload
        loadbalance
    }
kind: ConfigMap
metadata:
  creationTimestamp: "2023-07-12T12:51:47Z"
  name: coredns
  namespace: kube-system
  resourceVersion: "257"
  uid: d0dcf50e-f6c7-457b-804e-a333996b3042
```

IP do servidor

```
root@s2-master-1:~# k -n kube-system get svc | grep dns
kube-dns         ClusterIP   10.96.0.10       <none>        53/UDP,53/TCP,9153/TCP   33d
```

```
root@s2-master-1:~# k run temp --image=busybox --rm -it -- /bin/sh
If you don't see a command prompt, try pressing enter.
/ #
/ #
/ # cat /etc/resolv.conf
nameserver 10.96.0.10
search default.svc.cluster.local svc.cluster.local cluster.local
```

```
root@s2-master-1:~# grep clusterDNS -A1 /var/lib/kubelet/config.yaml
clusterDNS:
- 10.96.0.10
```

Redes no Kubernetes

RNP
ORGANIZAÇÃO SOCIAL DO MCTI
MINISTÉRIO DO TURISMO
MINISTÉRIO DA DEFESA
MINISTÉRIO DA SAÚDE
MINISTÉRIO DA EDUCAÇÃO
MINISTÉRIO DA CIÊNCIA, TECNOLOGIA E INOVAÇÕES
PÁTRIA AMADA BRASIL
GOVERNO FEDERAL