扫码加群技术交流

扫码报名成都meetup

HUAWEI

CloudNativeLives

Kubernetes网络模型原理剖析与实践

华为云容器团队核心架构师 & CNCF社区主要贡献者倾心打造

大 纲

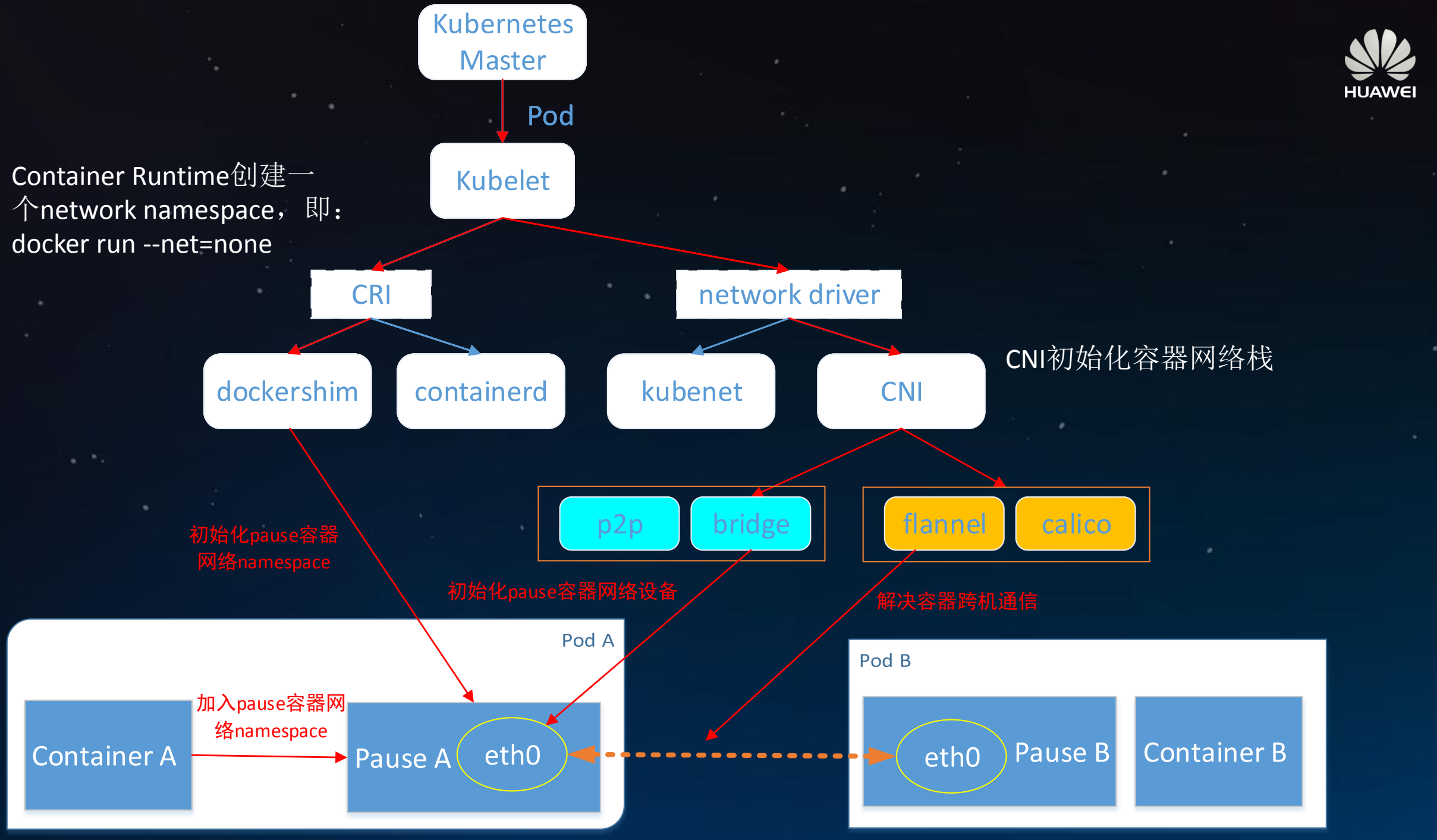
- **网络模型与CNI**
- Service
- Ingress
- DNS
- Network Policy

Kubernetes网络模型与CNI

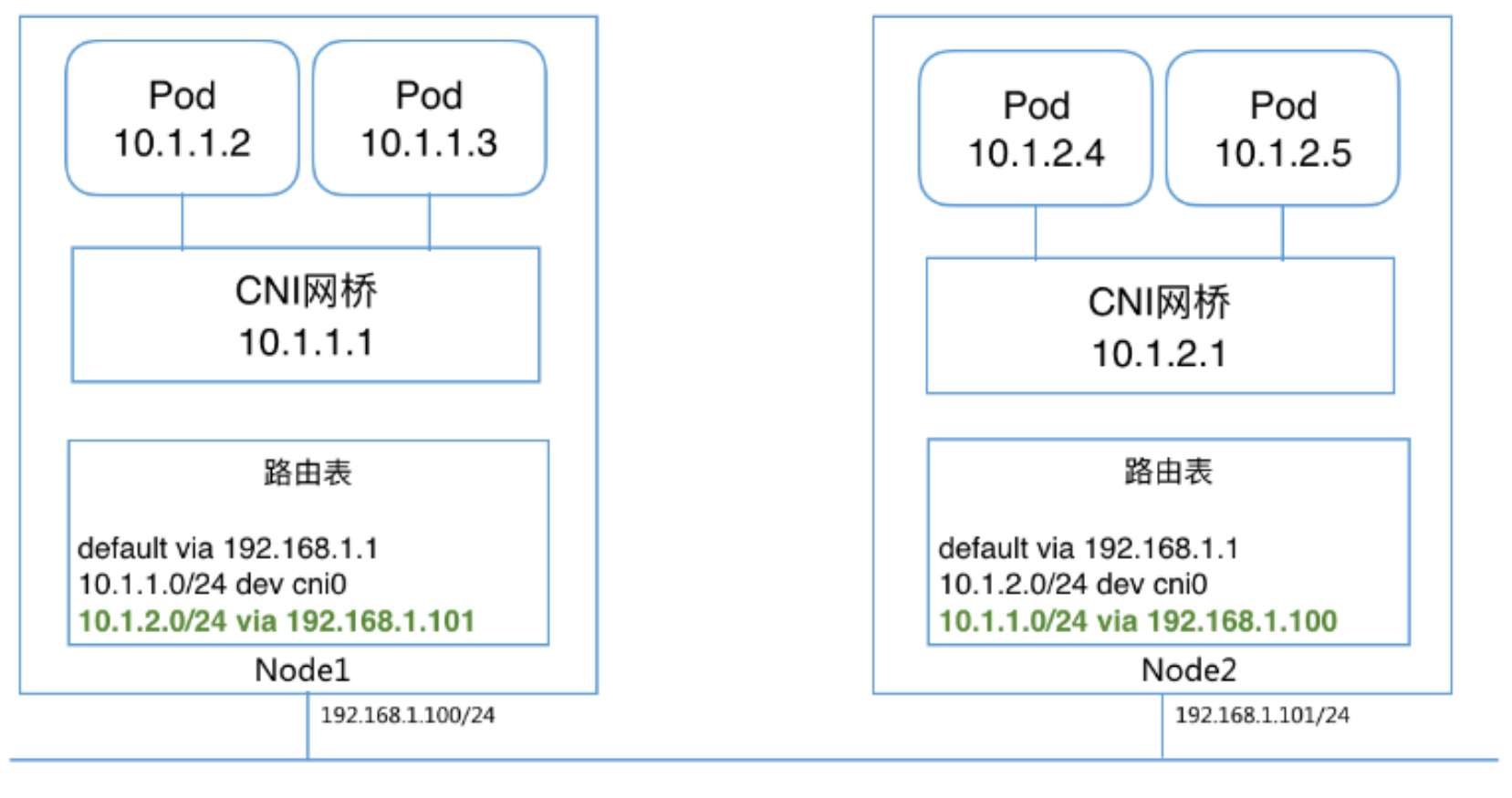
- 一个Pod一个IP
  - 每个Pod独立IP，Pod内所有容器共享网络namespace（同一个IP）
  - 容器之间直接通信，不需要NAT
  - Node和容器直接通信，不需要NAT
  - 其他容器和容器自身看到的IP是一样的

扁平网络：性能、可追溯、排错

- 集群内访问走Service，集群外访问走Ingress
- CNI（container network interface）用于配置Pod网络
  - 不支持docker网络


Kubernetes Master
Pod
Kubelet
Container Runtime创建一个network namespace，即：docker run --net=none
CRI
network driver
dockershim
containerd
kubenet
CNI
CNI初始化容器网络栈
p2p
bridge
flannel
calico
初始化pause容器网络namespace
初始化pause容器网络设备
解决容器跨机通信
Pod A
Container A
加入pause容器网络namespace
Pause A
eth0
Pod B
eth0
Pause B
Container B

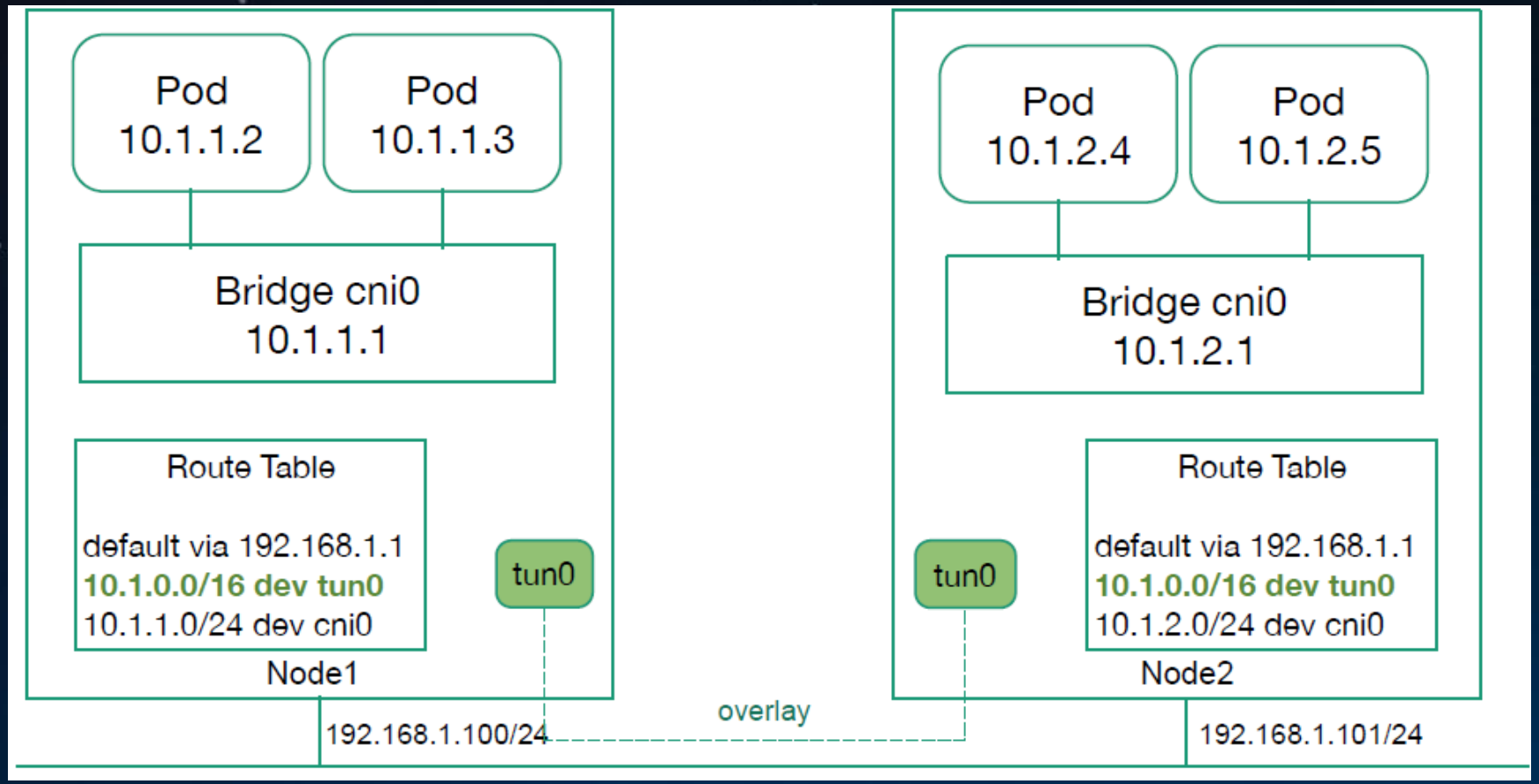
Bridge网络



注意：Bridge模式下，多主机网络通信需要额外配置主机路由，或使用overlay网络。

Overlay网络

Overlay是在物理网络之上的虚拟网络，VXLAN是当前最主流的Overlay标准
- VXLAN就是用UDP包头封装二层帧（MAC in UDP）

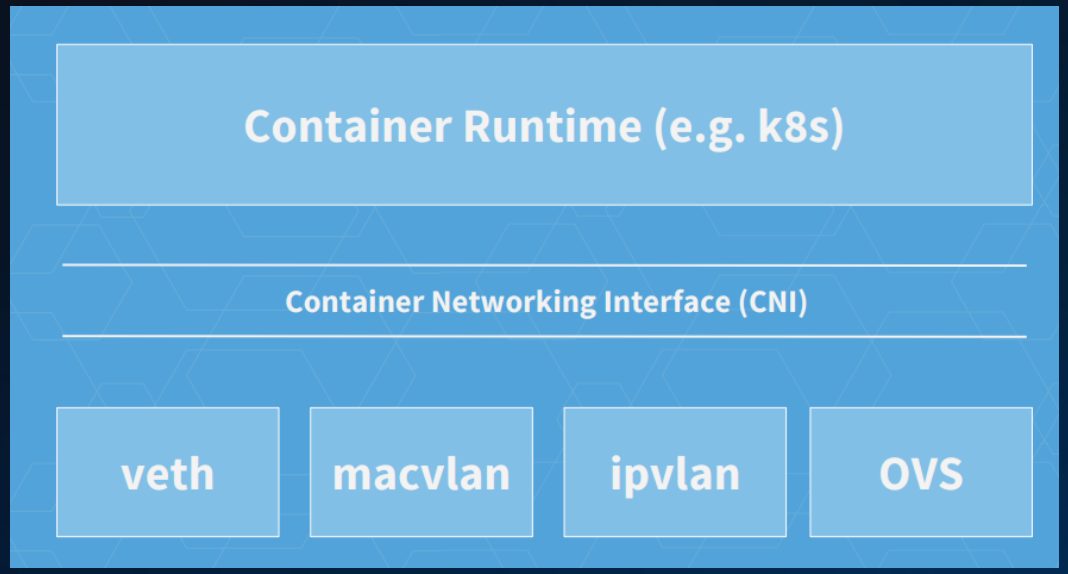


CNI: Container Network Interface

- 容器网络的标准化
- 使用JSON来描述网络配置
- 两类接口：
  - 配置网络 -- 创建容器时调用

    AddNetwork(net *NetworkConfig, rt* RuntimeConf) (types.Result, error)
  - 清理网络 -- 删除容器时调用

    DelNetwork(net *NetworkConfig, rt* RuntimeConf) error



- Bridge
- PTP
- IPVLAN
- MACVLAN
- VLAN
- PORTMAP

flannel
PROJECT CALICO
weaveworks
Google Cloud Platform
nuagenetworks
ROMANA
aws
Contiv
CNI-Genie
OPENCONTRAIL
OvS Open vSwitch
cilium
KUBE ROUTER

CNI插件：host-local + bridge

```
$ cat /etc/cni/net.d/10-mynet.conf
{
    "name": "mynet",
    "type": "bridge",
    "ipam": {
        "type": "host-local",
        "subnet": "10.10.0.0/16"
    }
}
```

CNI plugin二进制文件： /opt/cni/bin/{host-local, bridge…}

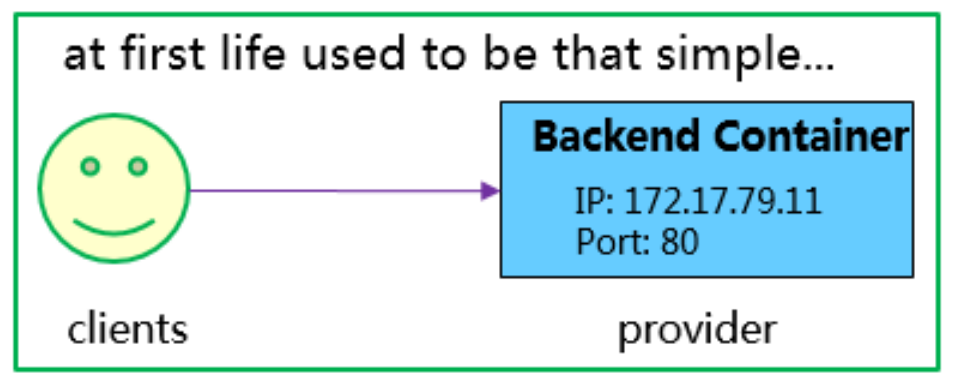
CNI插件：带宽控制

```
{
    "kind": "Pod",
    "metadata": {
        "name": "iperf-slow",
        "annotations": {
            "kubernetes.io/ingress-bandwidth": "10M",
            "kubernetes.io/egress-bandwidth": "10M"
        }
    }
}
```

```
{
 "type": "bandwidth",
 "runtimeConfig": {
  "trafficShaping": {
   “ingressRate”: “10000”
   "egressRate": “10000"
  }
 }
}
```

大 纲

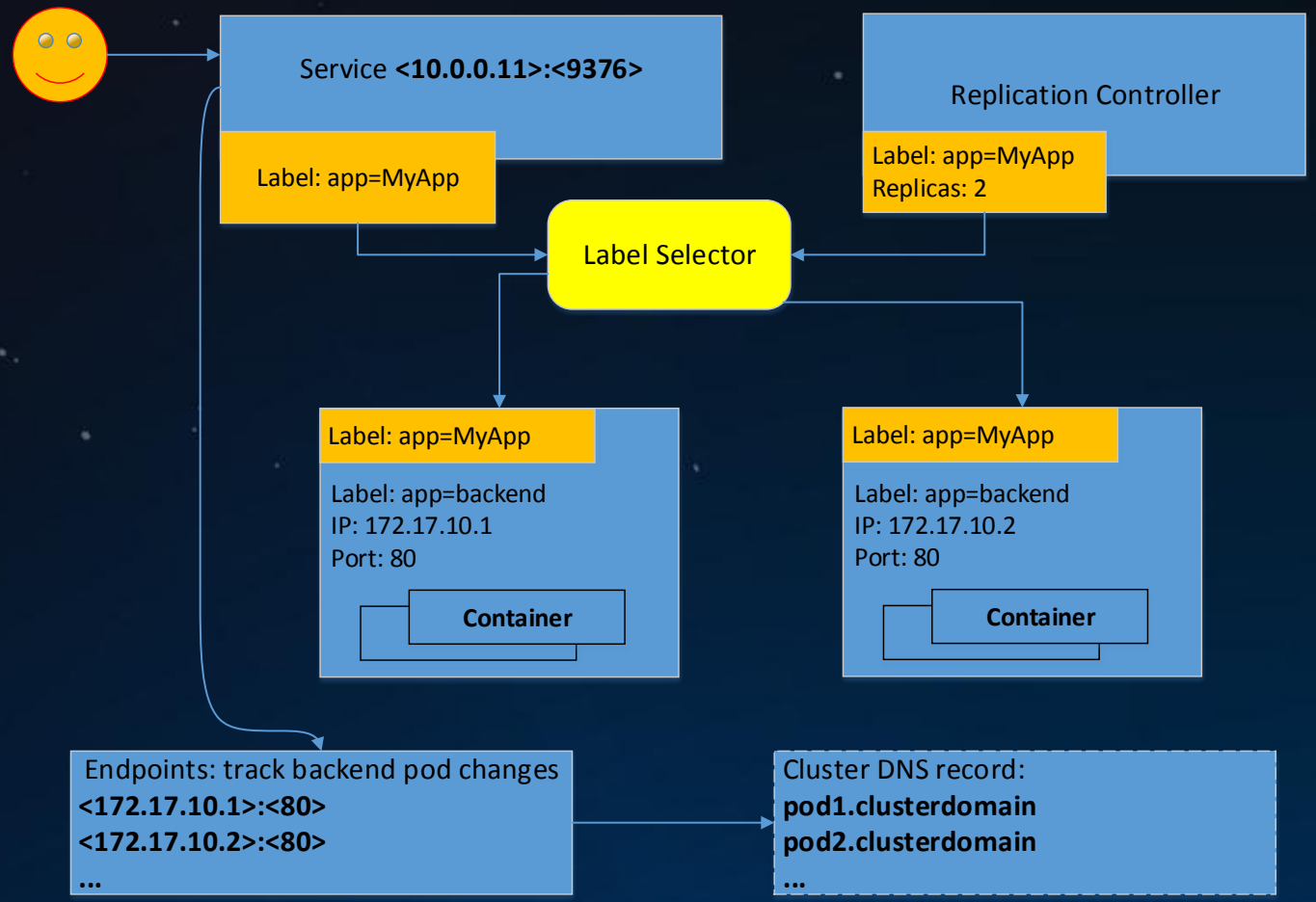
- 网络模型与CNI
- **Service**
- Ingress
- DNS
- Network Policy



但，简单的生活总是暂时的：

- 多个后端实例，如何做到负载均衡？
- 如何保持会话亲和性？
- 容器迁移，IP发生变化如何访问？
- 健康检查怎么做？
- 怎么通过域名访问？

Kubernetes Service

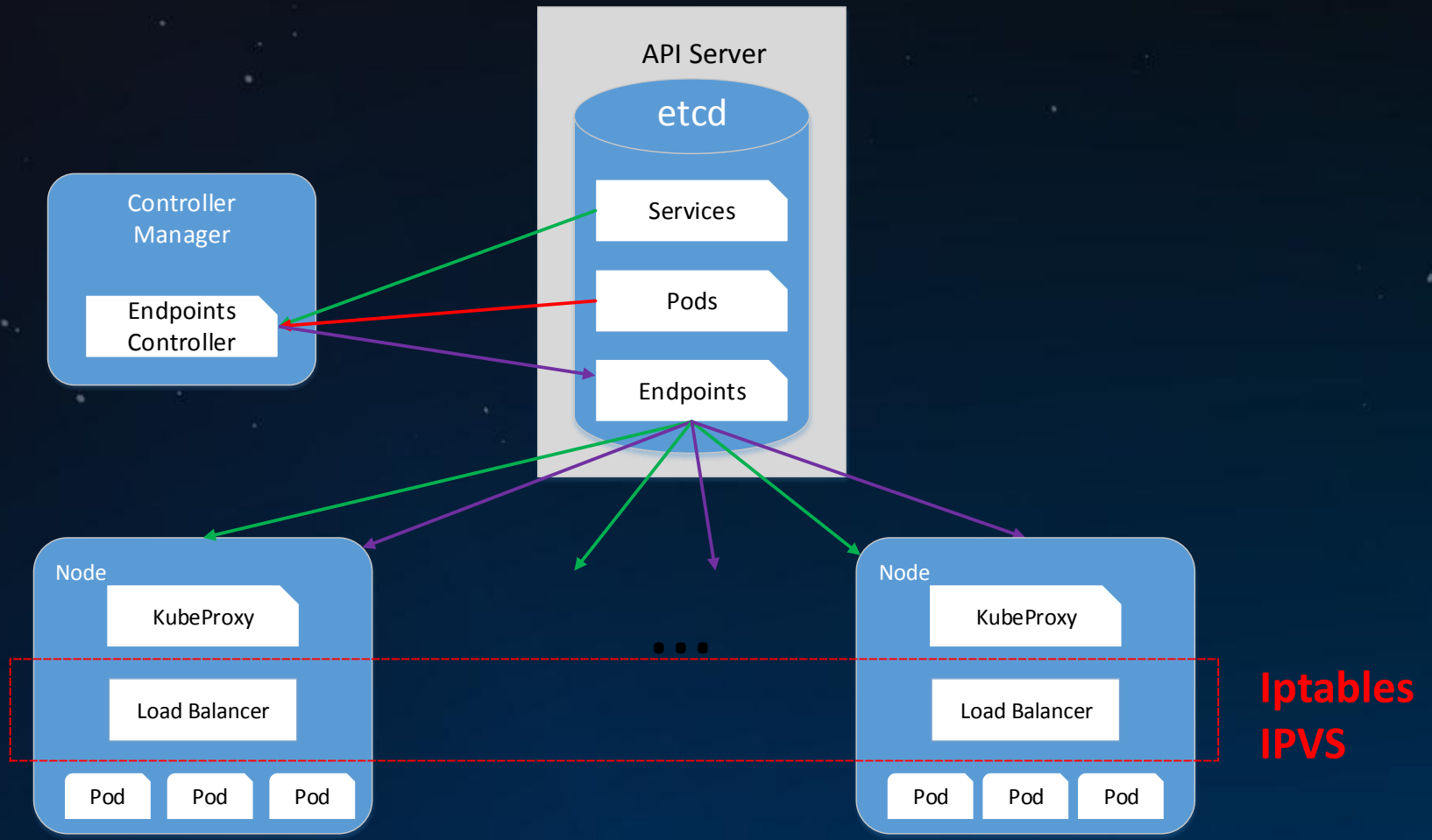


Service和Endpoints定义

```yaml
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
  namespace: default
spec:
  clusterIP: 10.101.28.148
  ports:
  - name: http
    port: 80
    protocol: TCP
    targetPort: 8080
  selector:
    app: nginx
```

```yaml
apiVersion: v1
kind: Endpoints
metadata:
  name: nginx-service
  namespace: default
subsets:
- addresses:
  - ip: 172.17.0.2
    nodeName: 100-106-179-237.node
    targetRef:
      kind: Pod
      name: nginx-rc-c8tw2
      namespace: default
  - ip: 172.17.0.3
    nodeName: 100-106-179-238.node
    targetRef:
      kind: Pod
      name: nginx-rc-x14tv
      namespace: default
  ports:
  - name: http
    port: 8080
    protocol: TCP
```

Service工作原理



Kubernetes Service类型

- ClusterIP
  - **默认类型**，自动分配集群内部可以访问的虚IP——Cluster IP。
- NodePort
  - 为Service在Kubernetes集群的每个Node上分配一个端口，即NodePort，集群内/外部可基于任何一个NodeIP:NodePort的形式来访问Service。
- LoadBalancer
  - 需要跑在特定的cloud provider上
  - Service Controller自动创建一个外部LB并配置安全组
  - 对集群内访问，kube-proxy用iptables或ipvs实现了云服务提供商LB的部分功能：L4转发，安全组规则等。

Kubernetes服务发现

- 环境变量
  - Kubelet为每个Pod注入所有Service的环境变量信息，形如：

```
REDIS_MASTER_SERVICE_HOST=10.0.0.11
REDIS_MASTER_SERVICE_PORT=6379
REDIS_MASTER_PORT=tcp://10.0.0.11:6379
REDIS_MASTER_PORT_6379_TCP=tcp://10.0.0.11:6379
REDIS_MASTER_PORT_6379_TCP_PROTO=tcp
REDIS_MASTER_PORT_6379_TCP_PORT=6379
REDIS_MASTER_PORT_6379_TCP_ADDR=10.0.0.11
```

缺点：环境变量洪泛，docker启动参数过长直接导致启动容器失败

- 域名

  假设Service（my-svc）在namespace（my-ns）中，暴露名为http的TCP端口：
  - A记录： **my-svc.my-ns → Cluster IP**
  - SRV记录：**_http._tcp.my-svc.my-ns → http端口号**

Service实现之iptables

Iptables是用户空间应用程序，通过配置Netfilter规则表（ Xtables ）来构建linux内核防火墙。

```
Chain PREROUTING (policy ACCEPT)
target     prot opt source               destination
KUBE-SERVICES  all  --  0.0.0.0/0            0.0.0.0/0

Chain KUBE-SERVICES (2 references)
target     prot opt source               destination
KUBE-SVC-6IM33IEVEEV7U3GP  tcp  --  0.0.0.0/0     10.20.30.40  tcp dpt:80

Chain KUBE-SVC-6IM33IEVEEV7U3GP (1 references)
target     prot opt source               destination
KUBE-SEP-Q3UCPZ54E6Q2R4UT  all  --  0.0.0.0/0            0.0.0.0/0

Chain KUBE-SEP-Q3UCPZ54E6Q2R4UT (1 references)
target     prot opt source               destination
DNAT       tcp  --  0.0.0.0/0            0.0.0.0/0       tcp to:172.17.0.2:8080
```

Service IP:Port -> PREROUTING(OUTPUT) -> KUBE-SERVICES -> KUBE-SVC-XXX -> KUBE-SEP-XXX -> Pod IP:Port

Service实现之IPVS

IPVS是LVS的负载均衡模块，亦基于netfilter，但比iptables性能更高。

```
Name:              nginx-service
Type:              ClusterIP
IP:                10.102.128.4
Port:              http    80/TCP
Endpoints:         10.244.0.235:8080,10.244.1.237:8080
Session Affinity:   10800

# ip addr
73: kube-ipvs0: <BROADCAST,NOARP> mtu 1500 qdisc noop state DOWN qlen 1000
    link/ether 1a:ce:f5:5f:c1:4d brd ff:ff:ff:ff:ff:ff
    inet 10.102.128.4/32 scope global kube-ipvs0
       valid_lft forever preferred_lft forever

# ipvsadm -ln
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Flags
  -> RemoteAddress:Port           Forward Weight ActiveConn InActConn
TCP  10.102.128.4:80 rr persistent 10800
  -> 10.244.0.235:8080            Masq    1      0          0
  -> 10.244.1.237:8080            Masq    1      0          0
```

iptables VS. IPVS

Service数数量	1	5000	20000
iptables规则数量	8	40000	160000
增加1条iptables规则时延	50 us	11 min	5 hours
增加1条IPVS规则	30 us	50 us	70 us

集群转发模式		内网访问		外网访问	
		吞吐量	平均时延	吞吐量	平均时延
500 并发	Iptables	23353/s	30.11ms	22030/s	30.14ms
	Ipvs	31094/s	30.06ms	27472/s	30.07ms
1000 并发	Iptables	28492/s	125.22ms	22029/s	350.29ms
	Ipvs	31361/s	30.16ms	26880/s	350.29ms

Metrics	Service数量	IPVS	iptables
内存消耗	1000	386 MB	1.1 G
	5000	N/A	1.9 G
	10000	542 MB	2.3 G
	15000	**N/A**	**OOM**
	50000	1272 MB	OOM
CPU使用率	1000	**0%**	N/A
	5000		50% - 85%
	10000		50%-100%
	15000		N/A
	50000		N/A

如何从集群外访问Kubernetes Service？

- 使用NodePort类型的Service
  - 要求Node有对外可访问IP
- 使用LoadBalancer类型的Service
  - 要求在特定的云服务上跑K8S

Service只提供L4负载均衡功能，而没有L7功能。

大 纲

- 网络模型与CNI
- Service
- **Ingress**
- DNS
- Network Policy

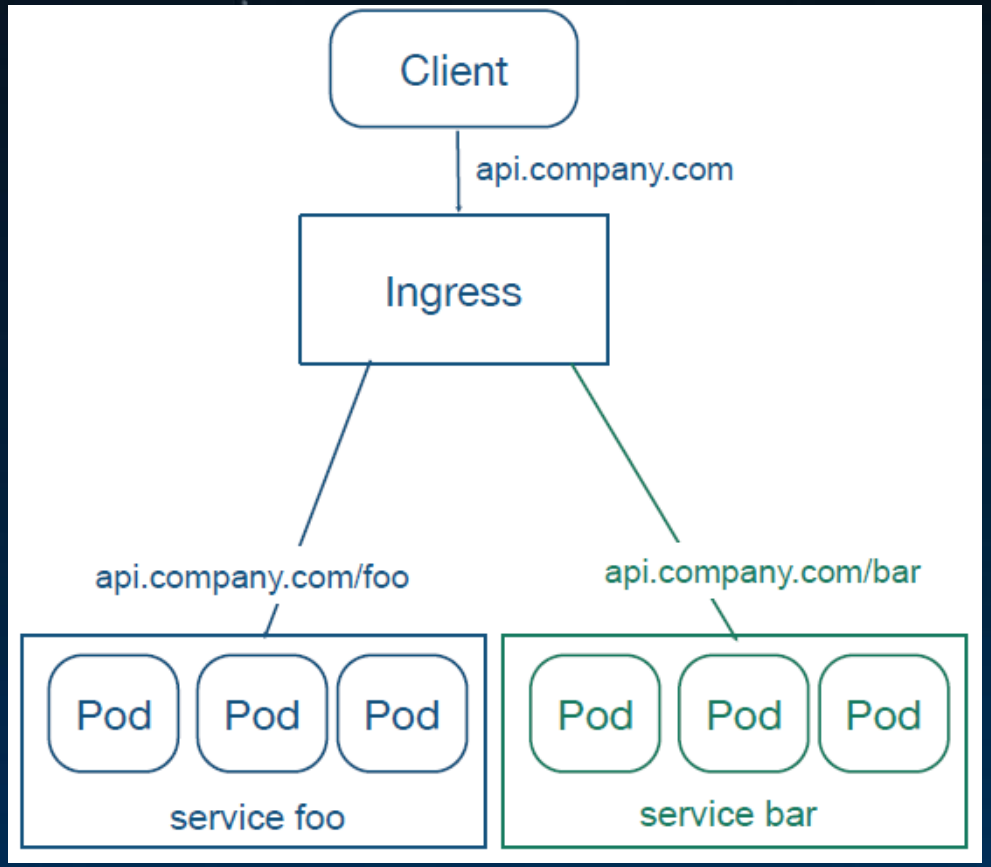
Ingress

Ingress是授权入站连接到达集群服务的规则集合

- 支持通过URL方式将Service暴露到K8S集群外，Service之上的L7访问入口
- 支持自定义Service的访问策略
- 提供按域名访问的虚拟主机功能
- 支持TLS

```
internet
    |
-----------
[ Services ]
```

```
internet
    |
[ Ingress ]
--|-----|--
[ Services ]
```



Ingress

```
apiVersion:
extensions/v1beta1
kind: Ingress
metadata:
  name: test-ingress
spec:
  tls:
  - secretName: testsecret
  backend:
    serviceName: testsvc
    servicePort: 80
```

```
$ kubectl get ing
NAME                RULE          BACKEND        ADDRESS
test-ingress        -             testsvc:80     107.178.254.228
```

ADDRESS：Ingress的访问入口地址，由Ingress Controller分配
BACKEND： K8S Service + Port
RULE： 自定义的访问策略。
若规则为空，则访问ADDRESS的所有流量都转发给BACKEND

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: test
spec:
  rules:
  - host: foo.bar.com
    http:
      paths:
      - path: /foo
        backend:
          serviceName: s1
          servicePort: 80
      - path: /bar
        backend:
          serviceName: s2
          servicePort: 80
```

当LB准备就绪时，Ingress Controller把填充ADDRESS字段

```
$ kubectl get ing
NAME      RULE              BACKEND    ADDRESS
test      -
          foo.bar.com
          /foo              s1:80
          /bar              s2:80
```

Ingress DIY

- 需要自己实现Ingress Controller
  - List/Watch K8S的Service，Endpoints，Ingress对象，刷新外部LB的规则和配置
  - 官方提供Nginx和GCE的Ingress Controller示例
- 想通过域名访问Ingress？
  - 需要自己配置域名和Ingress IP的映射：host文件，自己的DNS（不是Kube-dns）
- 嫌麻烦，懒得开发/配置？
  - Huawei CCE了解一下？ Ingress + 高性能ELB

大 纲

- 网络模型与CNI
- Service
- Ingress
- **DNS**
- **Network Policy**

Kubernetes DNS

- 解析Pod和Service的域名的，K8S集群内Pod使用
- Kube-dns和CoreDNS
- 对Service
  - A记录
    - 普通Service：my-svc.my-namespace.**svc.cluster.local → Cluster IP**
    - headless Service： my-svc.my-namespace.**svc.cluster.local → 后端Pod IP列表**
  - SRV记录：
    - _my-port-name._my-port-protocol.my-svc.my-namespace.svc.cluster.local → **Service Port**
- 对Pod
  - A记录
    - **pod-ip**.my-namespace.pod.cluster.local → Pod IP
  - 在Pod Spec指定hostname和subdomain
    - **hostname.subdomain**.my-namespace.pod.cluster.local → Pod IP

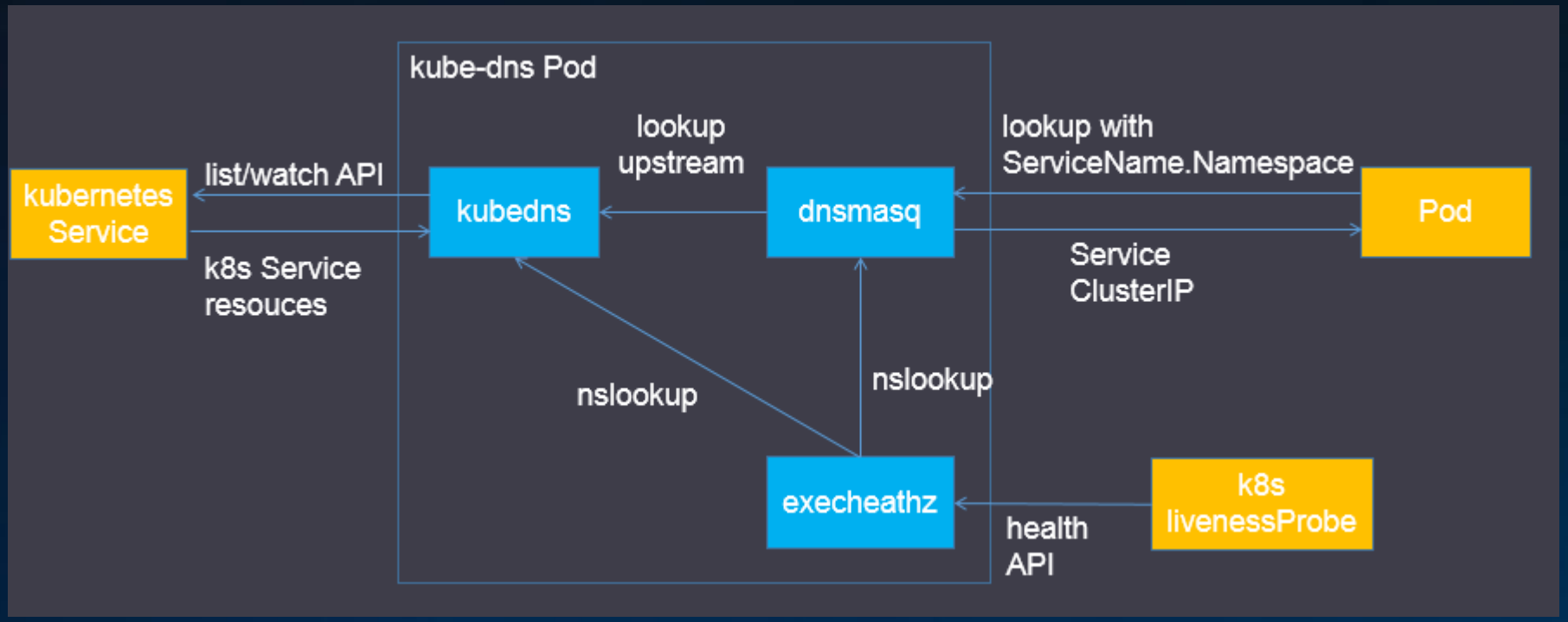
kubelet配置--cluster-dns把DNS的静态IP传递给每个容器

Kubelet传入--cluster-domain配置伪域名

1-2-3-4

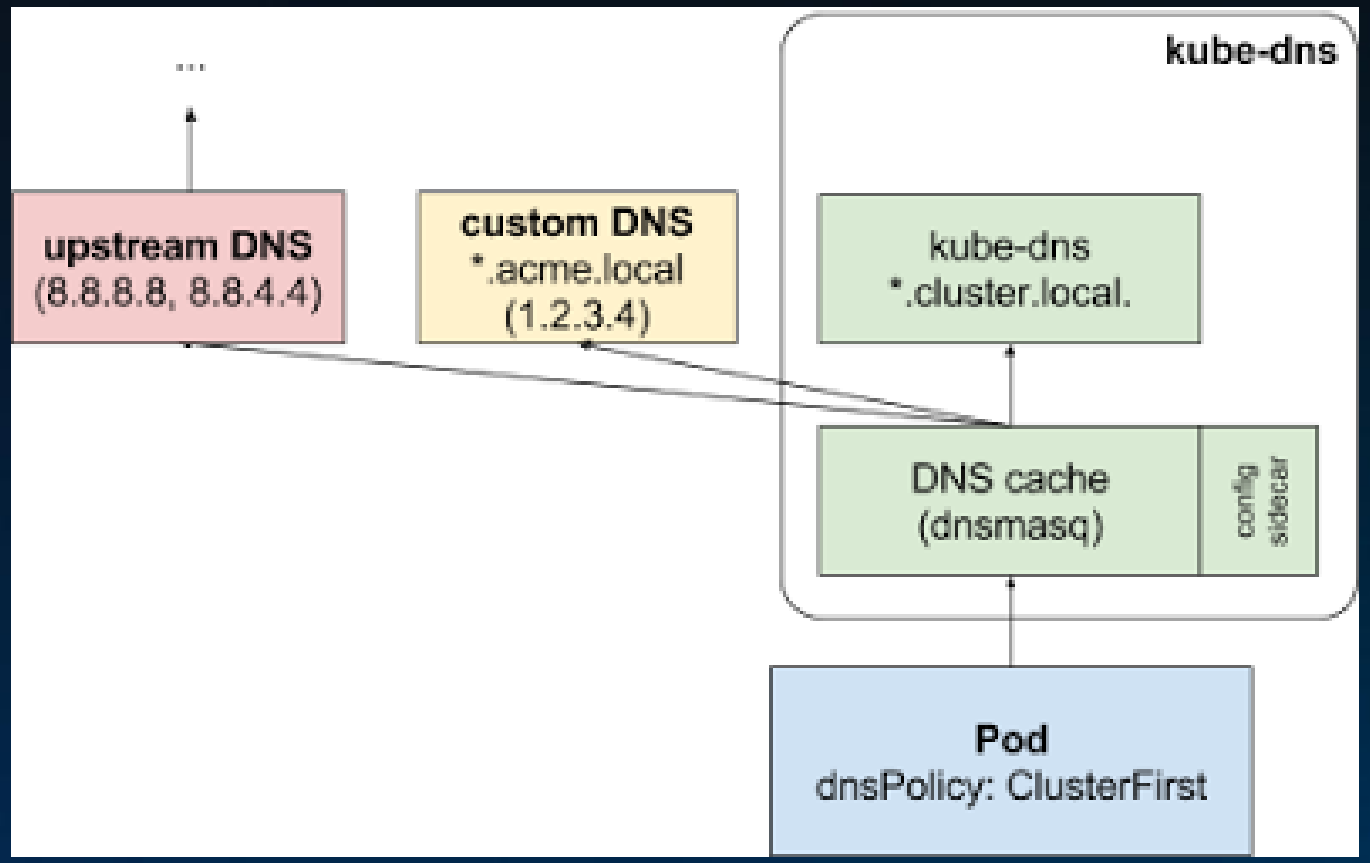
Kube-dns

kubedns：List/Watch K8S Service和Endpoints变化。接入SkyDNS，在内存中维护DNS记录，是dnsmasq的上游。
dnsmasq：DNS配置工具，监听53端口，为集群提供DNS查询服务。提供DNS缓存，降低kubedns压力。
sidecar：健康检查，检查kube-dns和dnsmasq的健康



Kube-dns级联查询

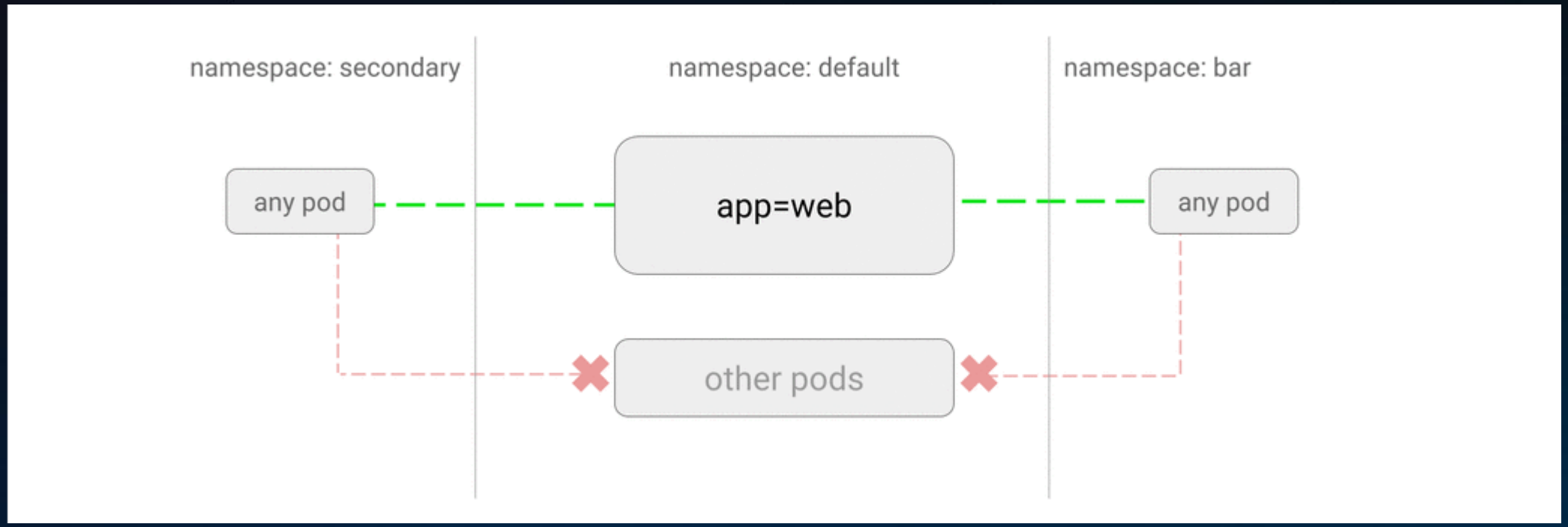
默认Pod会从Node继承DNS服务器配置，
也可以通过kubelet的--resolv-conf配置



大 纲

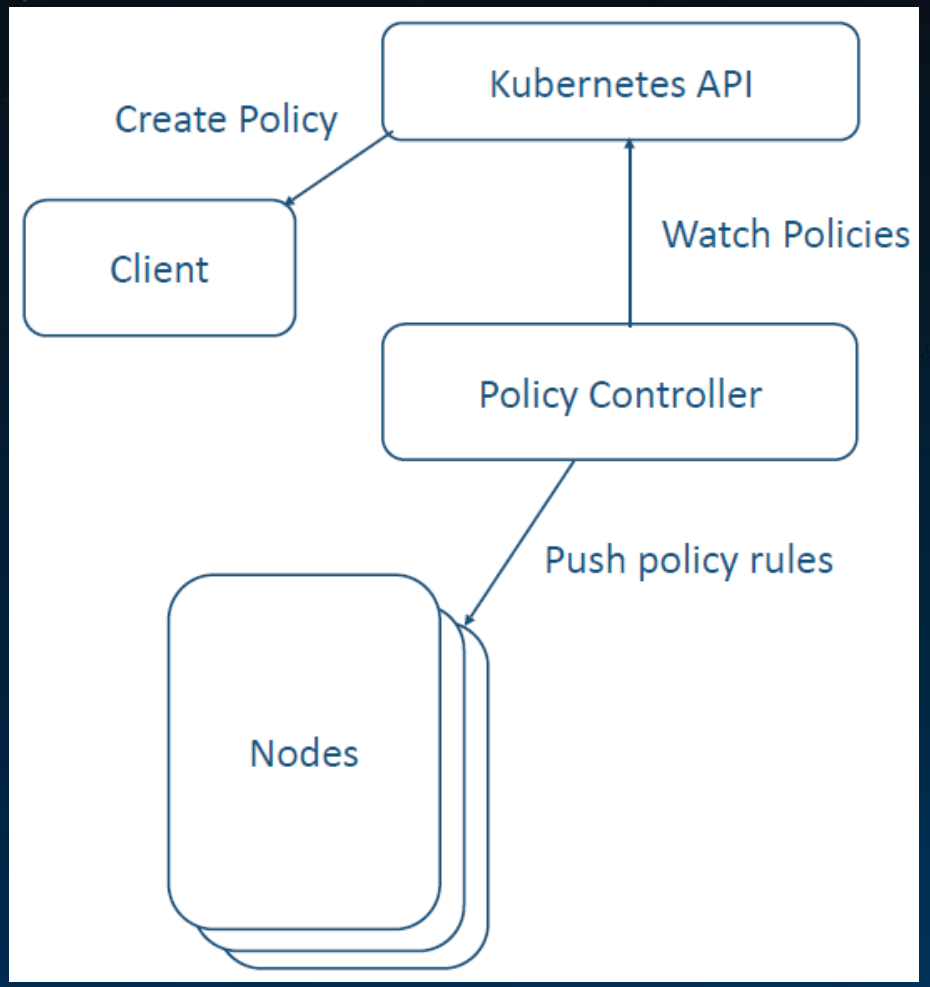
- 网络模型与CNI
- Service
- Ingress
- DNS
- **Network Policy**

Kubernetes的网络隔离



Network Policy是什么

- 基于源IP的访问控制列表
  - 限制Pod的进/出流量
  - 白名单
- Pod网络隔离的一层抽象
  - label selector
  - namespace selector
  - port
  - CIDR
- 没有Network Policy：“全网通”
- 网络插件实现Policy Controller



默认Network Policy

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: default-deny
spec:
  podSelector: {}
  policyTypes:
  - Ingress
  - Egress
```

Deny all ingress and egress

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-all
spec:
  podSelector: {}
  ingress:
  - {}
```

Allow all ingress

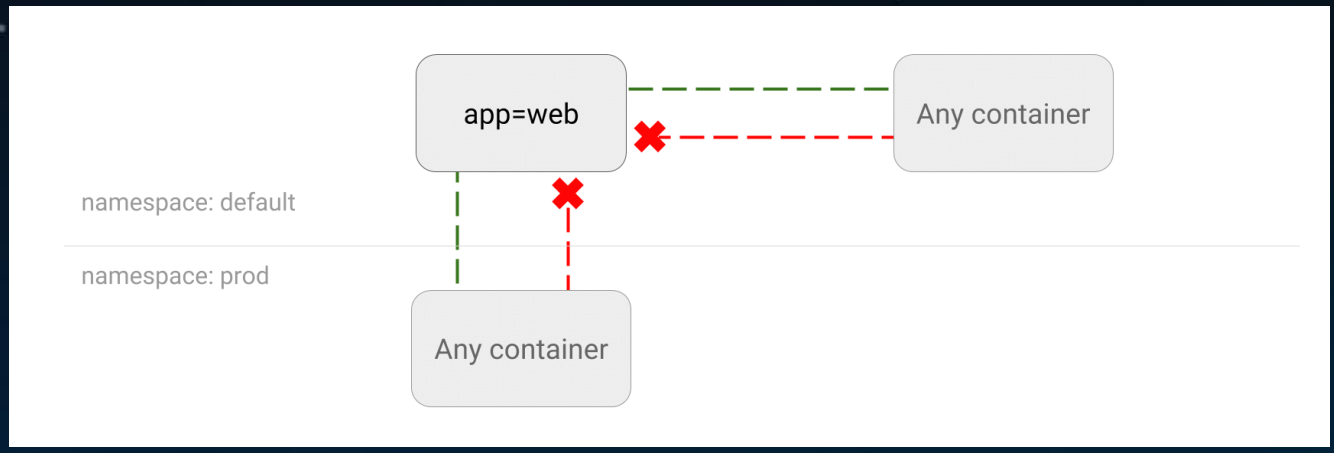
```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-all
spec:
  podSelector: {}
  egress:
  - {}
```

Allow all egress

注：{}代表允许所有，[]代表拒绝所有

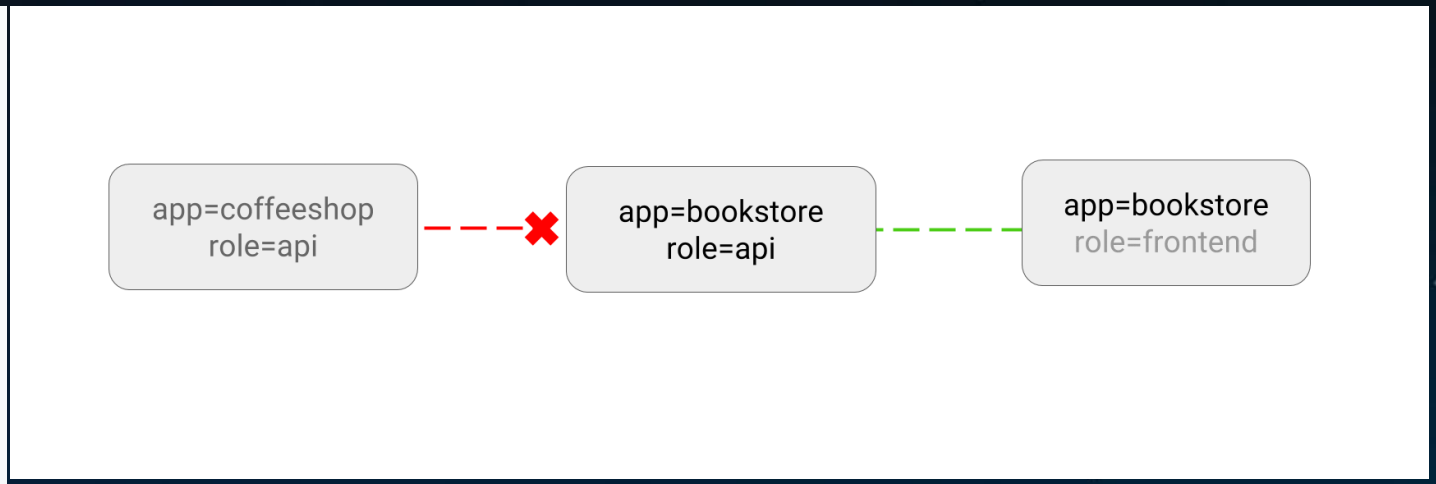
拒绝所有流量进入

```yaml
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: web-deny-all
spec:
  podSelector:
    matchLabels:
      app: web
  ingress: []
```



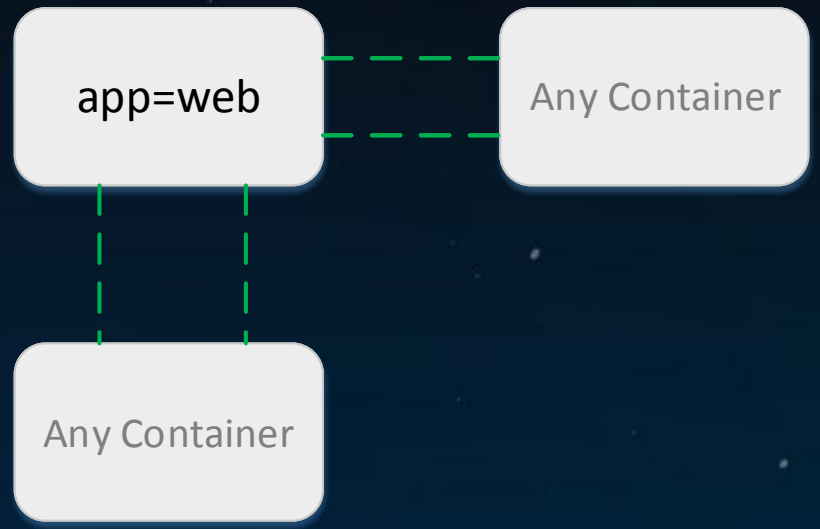
限制部分流量进入

```yaml
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: api-allow
spec:
  podSelector:
    matchLabels:
      app: bookstore
      role: api
  ingress:
  - from:
      - podSelector:
          matchLabels:
            app: bookstore
```



允许所有流量进入

```yaml
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: web-allow-all
  namespace: default
spec:
  podSelector:
    matchLabels:
      app: web
  ingress:
  - {}
```

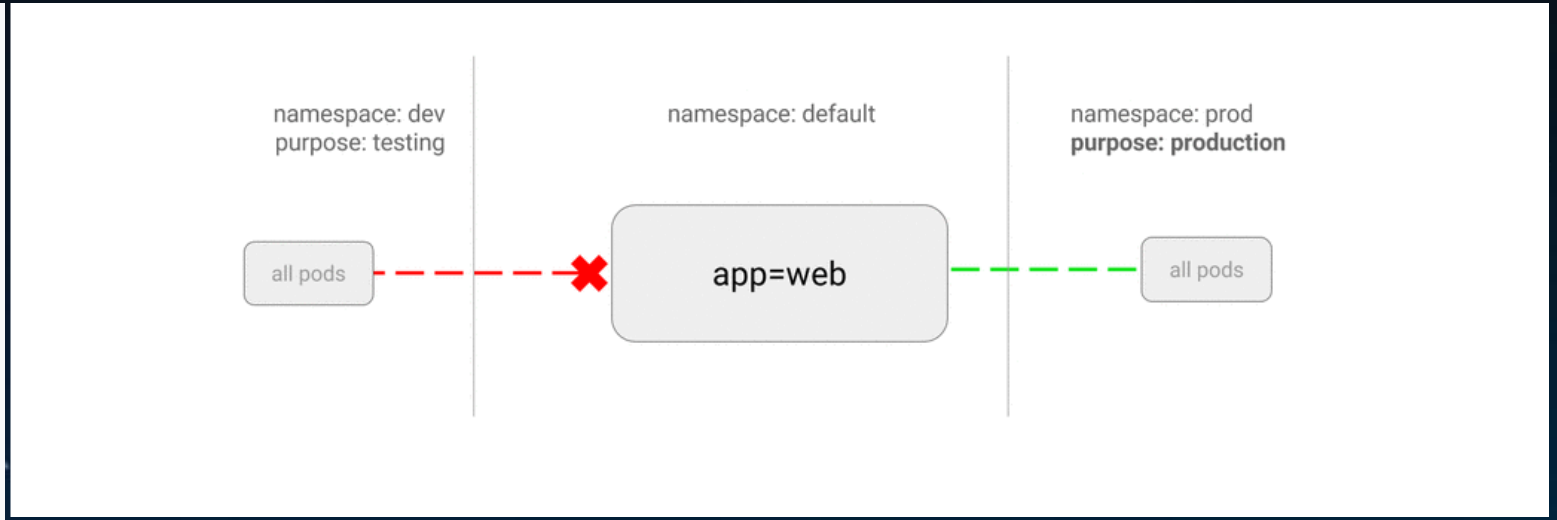


等价于：

ingress
- from
  podSelector: {}
  namespaceSelector: {}

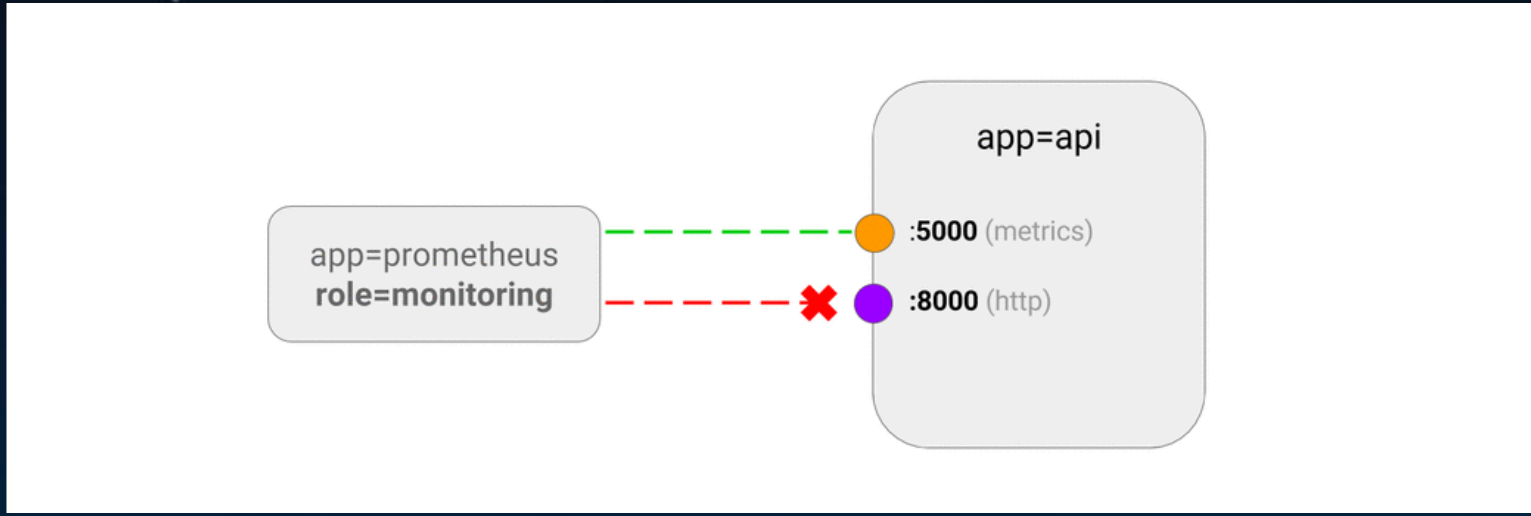
允许特定namespace的Pod流量进入

```yaml
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: web-allow-prod
spec:
  podSelector:
    matchLabels:
      app: web
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          purpose: production
```



限制流量从指定端口进入

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: api-allow-5000
spec:
  podSelector:
    matchLabels:
      app: api
  ingress:
  - ports:
    - port: 5000
    from:
    - podSelector:
        matchLabels:
          role: monitoring
```



支持Network Policy的网络插件

- Calico
- Cilium
- Weave Net
- Kube-router
- Romana

注意：flannel和kubenet不支持network policy

HUAWEI

Thank You

http://zhibo.huaweicloud.com/watch/2174406

直播 每周四 晚20:00

扫码加群技术交流

扫码报名成都meetup