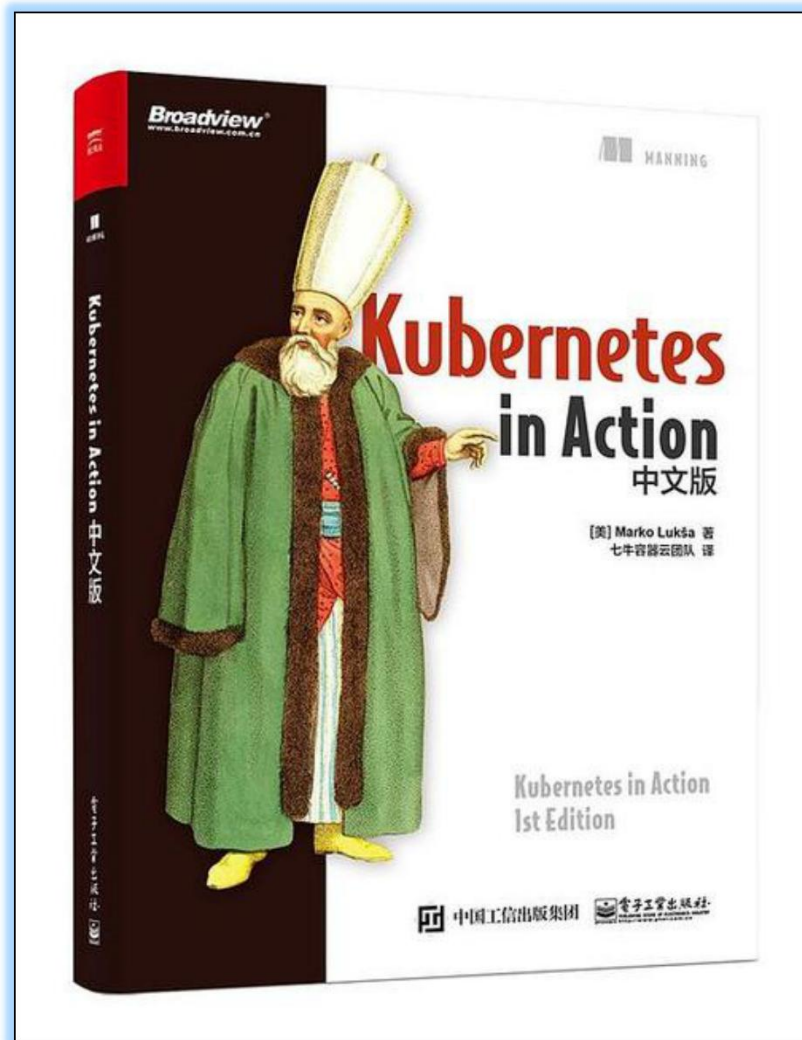


K8S IN ACTION

——5

服务：让客户端发现
pod并与之通信

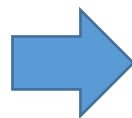
2020年5月



1. 介绍

□ pod会水平伸缩

——副本数不固定



屏蔽pod数量、
分布、ip，提供
固定的访问方式



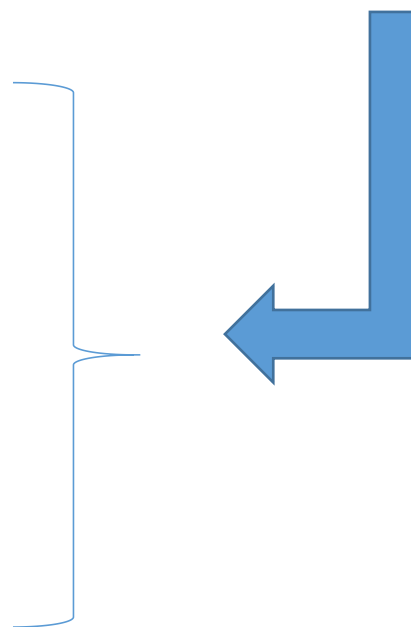
service

□ pod会销毁、漂移、重建

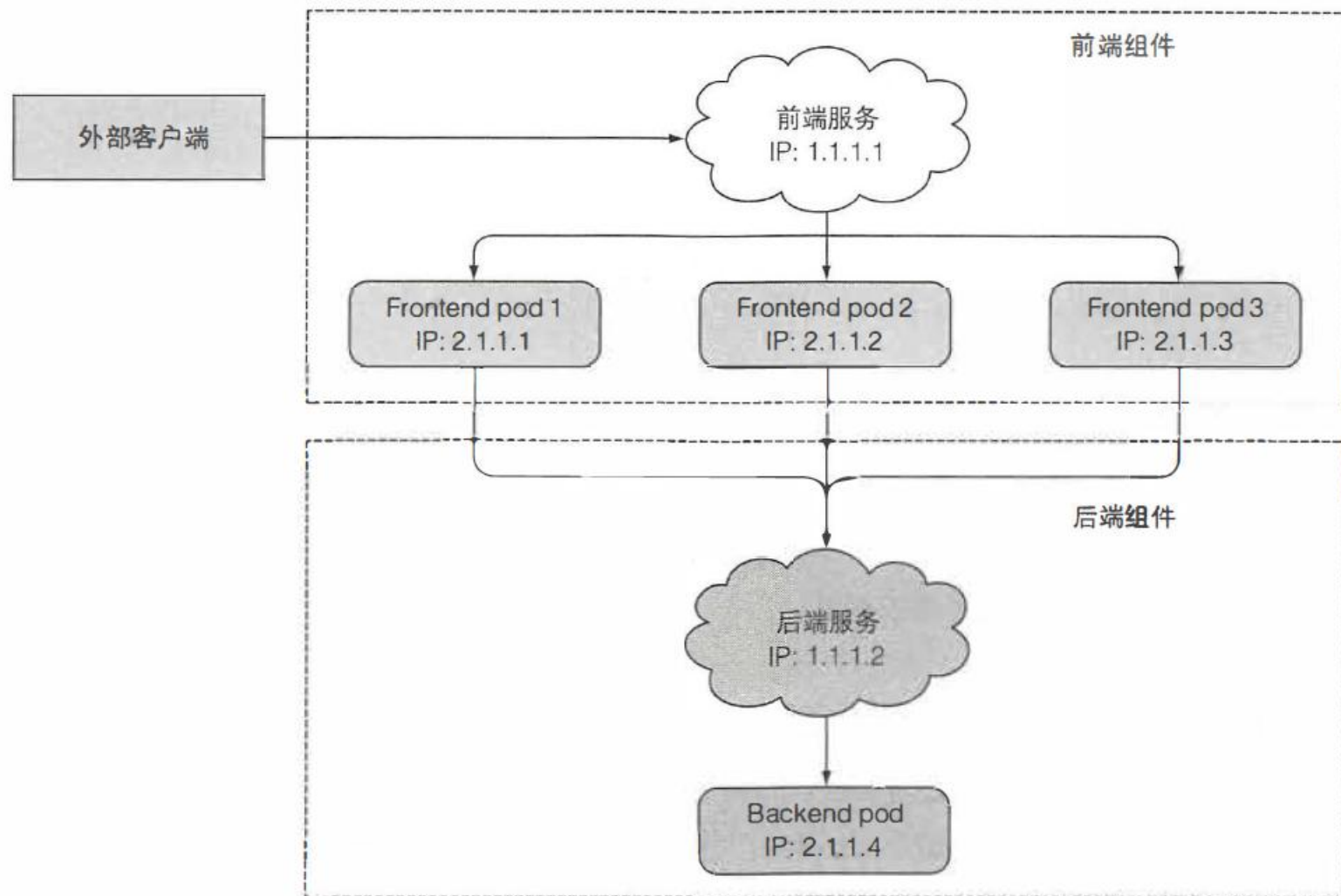
——IP不固定

提供稳定的一种稳定的访问方式
(外部->内部，内部->内部,内部->外部)

提供负载均衡



1. 介绍



1. 介绍

```
apiVersion: v1
kind: Service
metadata:
  name: kuba
spec:
  ports:
  - port: 80
    targetPort: 8080
  selector:
    app: kuba
```

该服务的可用端口

务将连接转发到的
容器端口

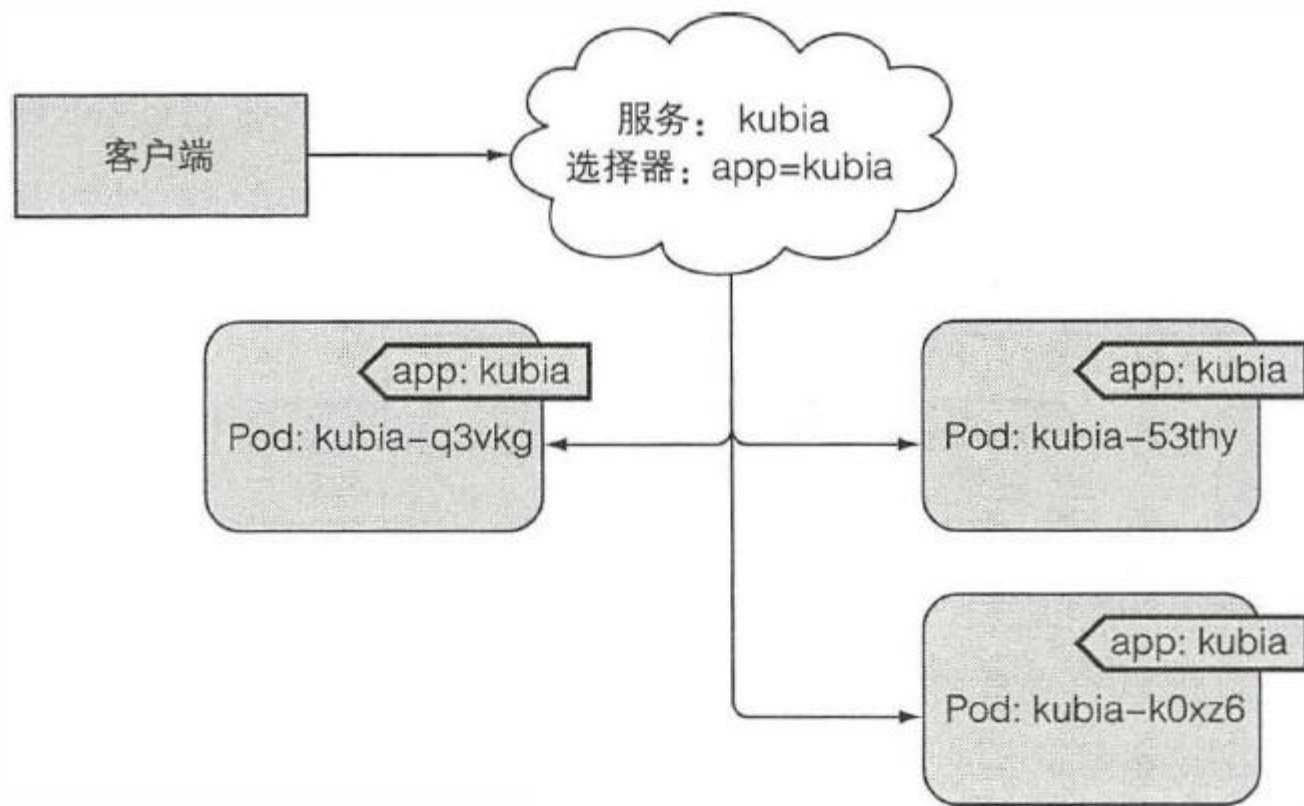
具有 app=kuba 签
的 pod 都 于该 务

域名访问
(集群内部)

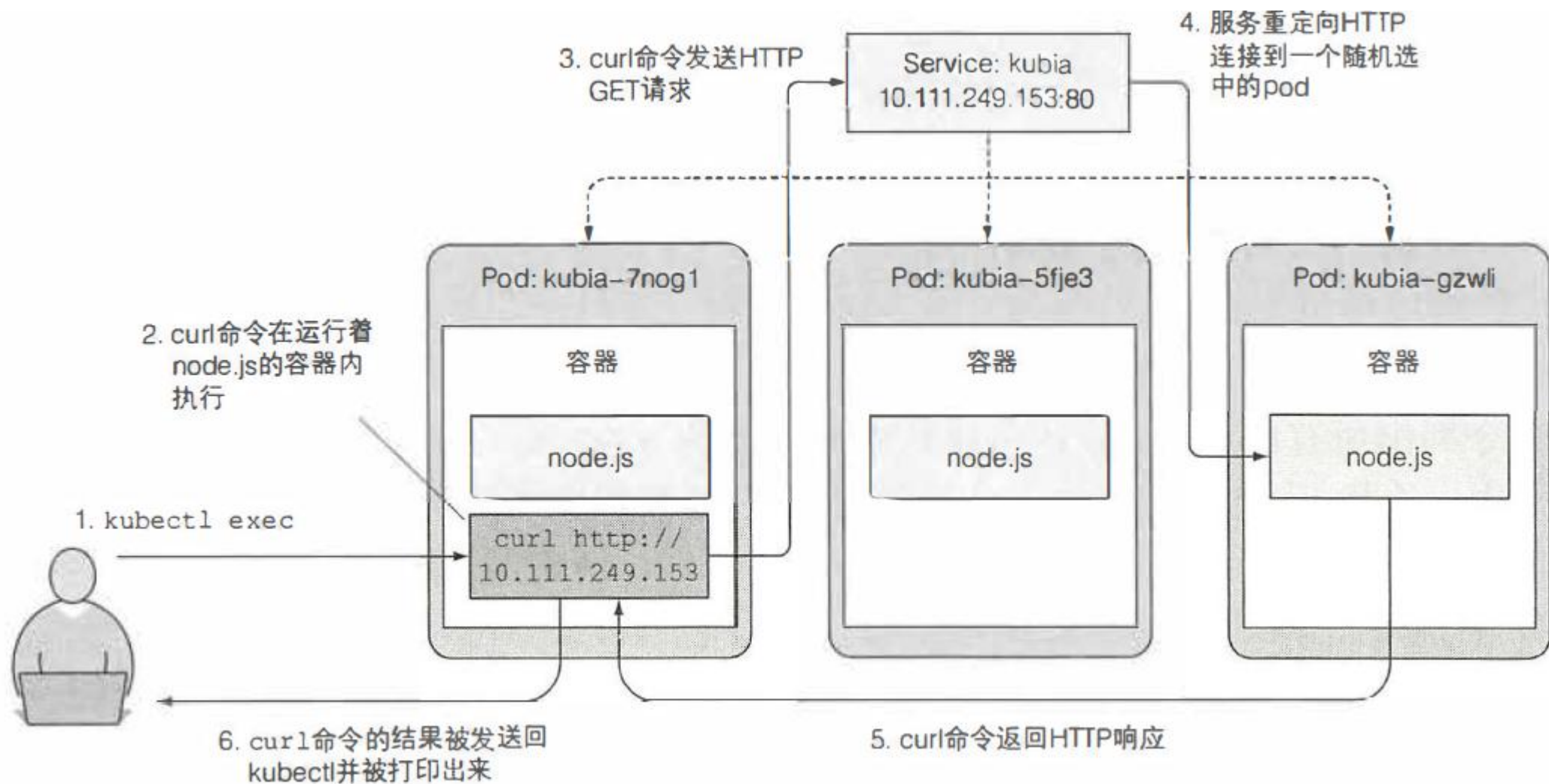
```
$ kubectl get svc
```

NAME	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
kubernetes	10.111.240.1	<none>	443/TCP	30d
kuba	10.111.249.153	<none>	80/TCP	6m

服务ip, 集
群内部可用



1. 介绍



1. 介绍 ——会话亲和性

```
apiVersion: v1
kind: Service
spec:
```

```
  sessionAffinity: ClientIP
```

```
  ...
```

工作在四层 (TCP/UDP)

支持两种类型

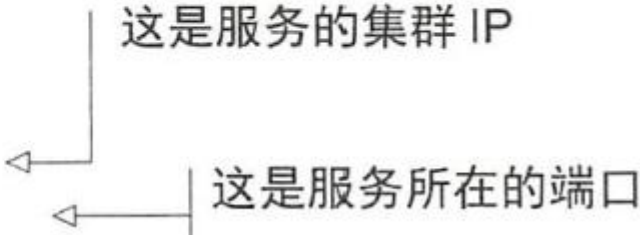
None : 默认, 不配置会话亲和性

ClientIP : 将同一个源IP
的包打到同一个pod (慎用)

1. 介绍 —— 集群内部pod的服务发现

1. 通过环境变量发现（服务早于客户端pod创建，同一命名空间下）

```
$ kubectl exec kubia-3inly env
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
HOSTNAME=kubia-3inly
KUBERNETES_SERVICE_HOST=10.111.240.1
KUBERNETES_SERVICE_PORT=443
...
KUBIA_SERVICE_HOST=10.111.249.153
KUBIA_SERVICE_PORT=80
```



这是服务的集群 IP

这是服务所在的端口

2. 通过dns发现（根据pod 中spec 的dnsPolicy 属性决定是否使用dns服务）

域名访问格式：服务名.命名空间.svc.cluster.local:端口

2. 连接集群外部服务 ——endpoint介绍

```
$ kubectl describe svc kuba
```

```
Name:          kuba
Namespace:     default
Labels:        <none>
Selector:      app=kuba
Type:          ClusterIP
IP:            10.111.249.153
Port:          <u^nset> 80/TCP
Endpoints:     10.108.1.4:8080,10.108.2.5:8080,10.108.2.6:8080
Session Affinity: None
No events.
```

用于创建 endpoint
列表的服务 pod 选
择器

代表服务
endpoint 的
pod 的 IP 和
端口列表

Endpoint 资源就是暴露一个服务的IP 地址和端口的列表

2. 连接集群外部服务 —— 创建没有选择器的服务

```
apiVersion: v1
kind: Service
metadata:
  name: external-service
spec:
  ports:
  - port: 80
```

服务的名字必须和
Endpoint 对象的名
字相匹配

服务中没有定义选
择器

定义一个名为 external-service 的服务，它将接收端口 80 上的传入连接。
并没有为服务定义一个 pod 选择器。

2. 连接集群外部服务 —— 创建endpoint资源

```
apiVersion: v1
kind: Endpoints
metadata:
  name: external-service
subsets:
  - addresses:
    - ip: 11.11.11.11
    - ip: 22.22.22.22
  ports:
    - port: 80
```

Endpoint 的名称必须和
服务的名称相匹配（见
之前的代码清单）

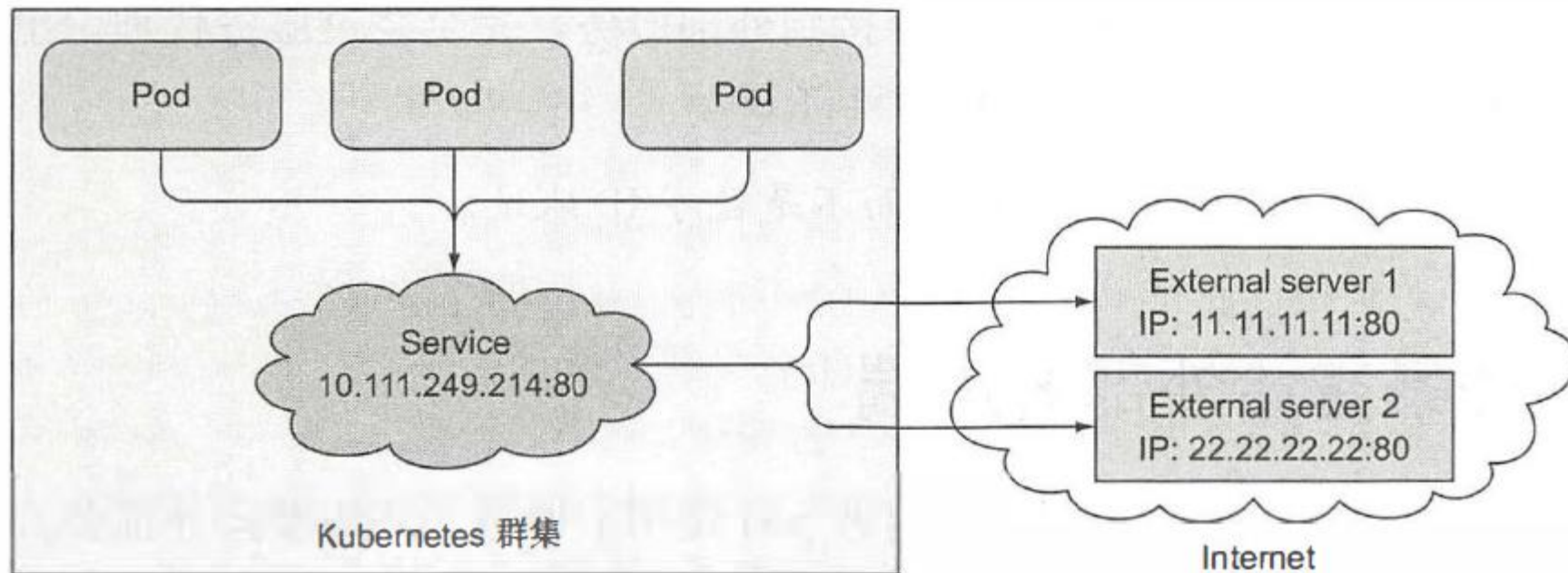


服务将连接重定向到 endpoint
的 IP 地址

← endpoint 的目标端口



2. 连接集群外部服务 —— pod访问外部服务



2. 连接集群外部服务 —— 通过其完全限定域名(FQDN)访问外部服务

```
apiVersion: v1
kind: Service
metadata:
  name: external-service
spec:
  type: ExternalName
  externalName: someapi.somecompany.com
  ports:
  - port: 80
```

代码的 type 被设置成
ExternalName

实际服务的完全限
定域名

1. 需要DNS支持
2. 服务创建完成后，pod通过external-service.default.SVC.cluster.local域名连接到外部服务

3. 服务暴露给集群外 —— 三种方式

- ◆ NodePort: 在集群每个节点都开一个端口，应对集群内的一个服务，外部通过 NodeIP:NodePort 访问服务
- ◆ LoadBalance: 由Kubernetes运行的云基础设施提供的专用负载均衡器（Load Balancer）将流量重定向到所有节点的节点端口，通过负载均衡器的IP连接服务
- ◆ Ingress服务: 通过一个IP地址公开多个服务，运行在HTTP层

3. 服务暴露给集群外 ——NodePort

```
apiVersion: v1
kind: Service
metadata:
  name: kubia-nodeport
spec:
  type: NodePort
  ports:
    - port: 80
      targetPort: 8080
      nodePort: 30123
  selector:
    app: kubia
```

为 NodePort 设置服务
类型

服务集群 IP 的端口号

背后 pod 的目标端
口号

通过集群节点的 30123 端
口可以访问该服务

通过所有nodeip
均可访问

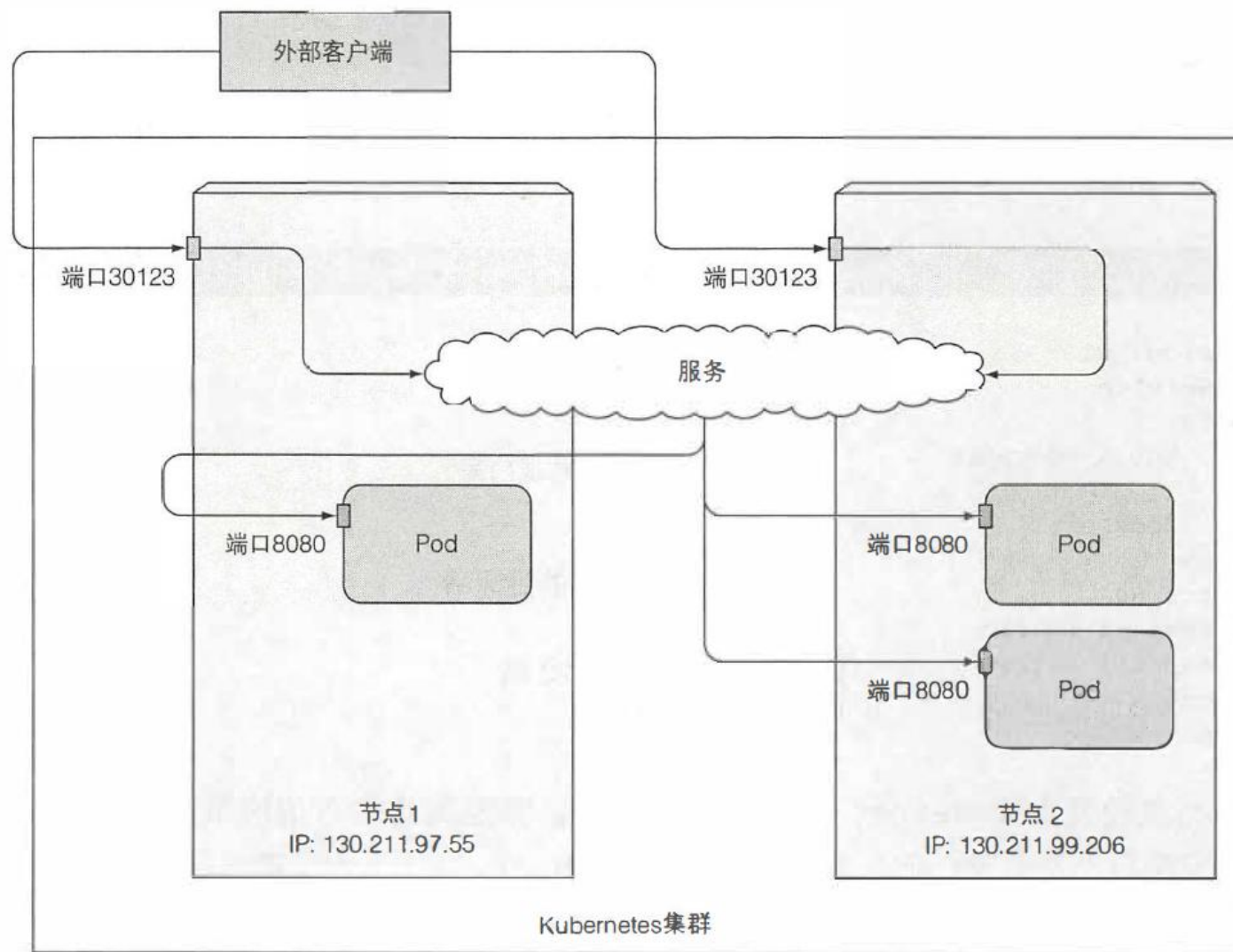
集群IP端口:节点IP端
口/访问协议

```
$ kubectl get svc kubia-nodeport
```

NAME	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
kubia-nodeport	10.111.254.223	<nodes>	80:30123/TCP	2m

- 10.11.254.223:80
- <1stnode'sIP>:30123
- <2ndnode'sIP>:30123

3. 服务暴露给集群外 ——NodePort



3. 服务暴露给集群外 ——LoadBalance

```
apiVersion: v1
kind: Service
metadata:
  name: kubia-loadbalancer
spec:
  type: LoadBalancer
  ports:
    - port: 80
      targetPort: 8080
  selector:
    app: kubia
```

该服务从 Kubernetes 集群的基础架构获取负载均衡器

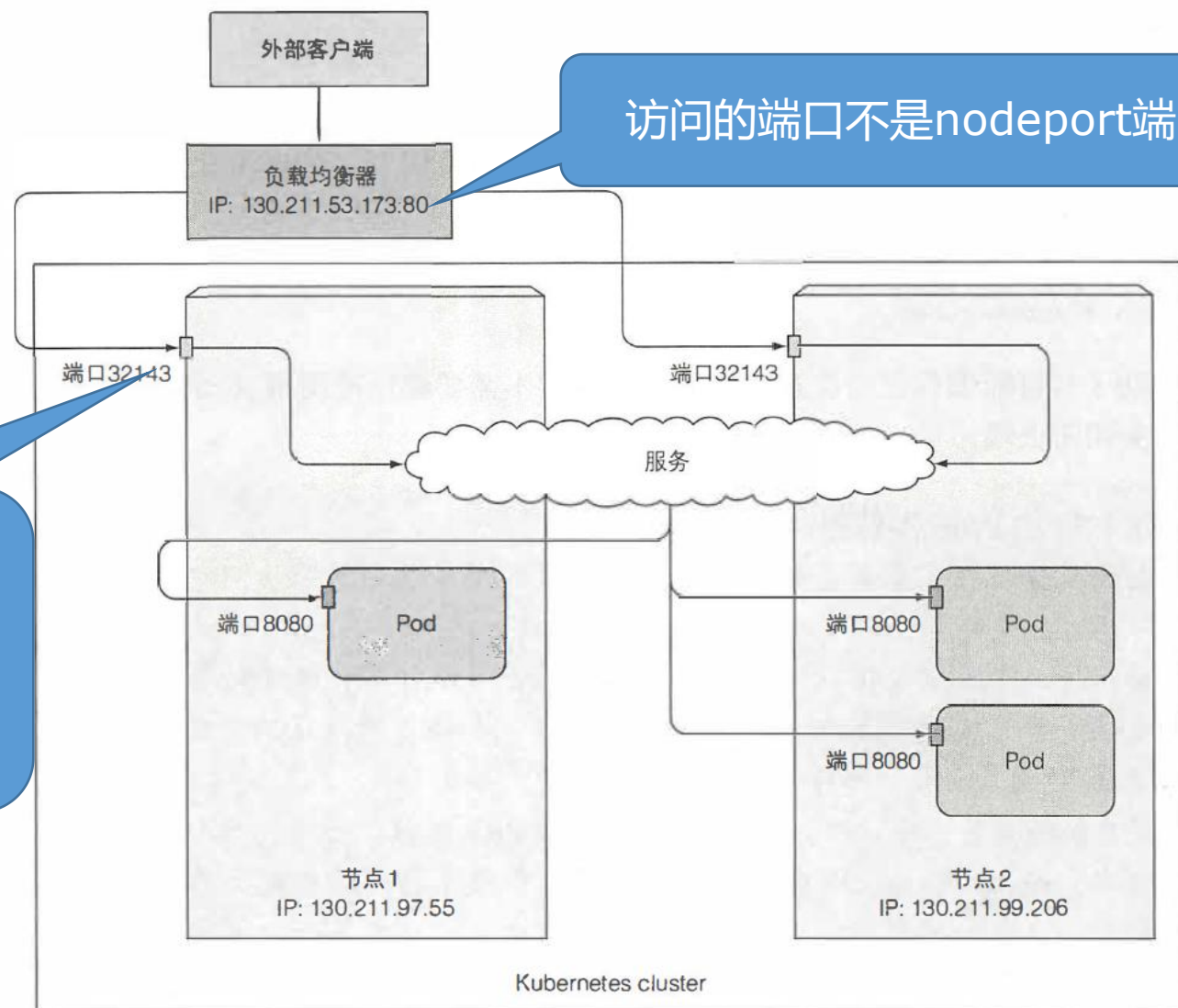
如果没有指定特定的节点端口，Kubernetes 将会选择一个端口。

该IP由云基础设施自动创建

```
$ kubectl get svc kubia-loadbalancer
```

NAME	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
kubia-loadbalancer	10.111.241.153	130.211.53.173	80:32143/TCP	1m

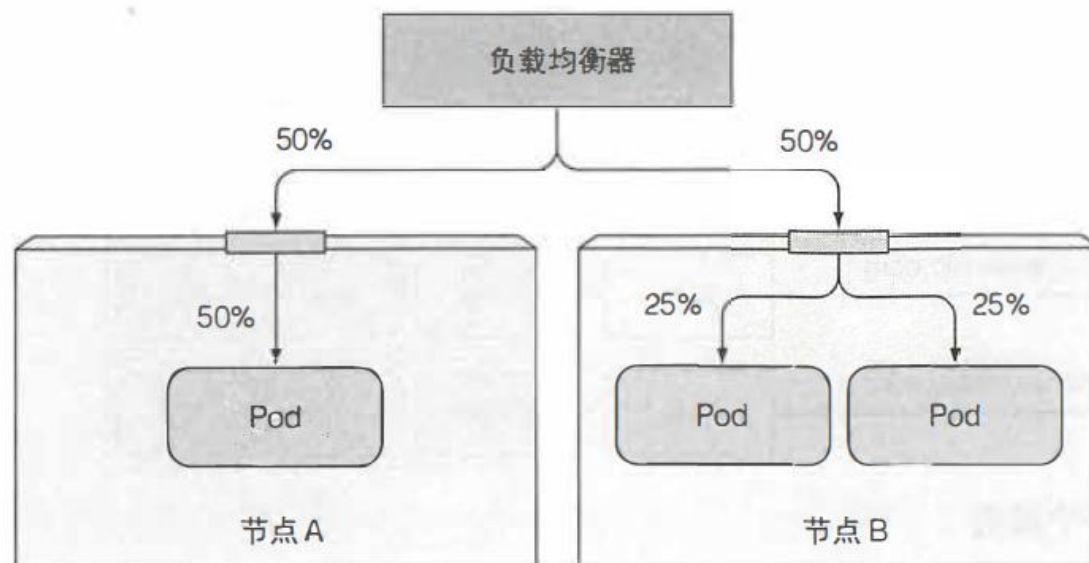
3. 服务暴露给集群外 —— LoadBalance



3. 服务暴露给集群外 ——减少额外跳数

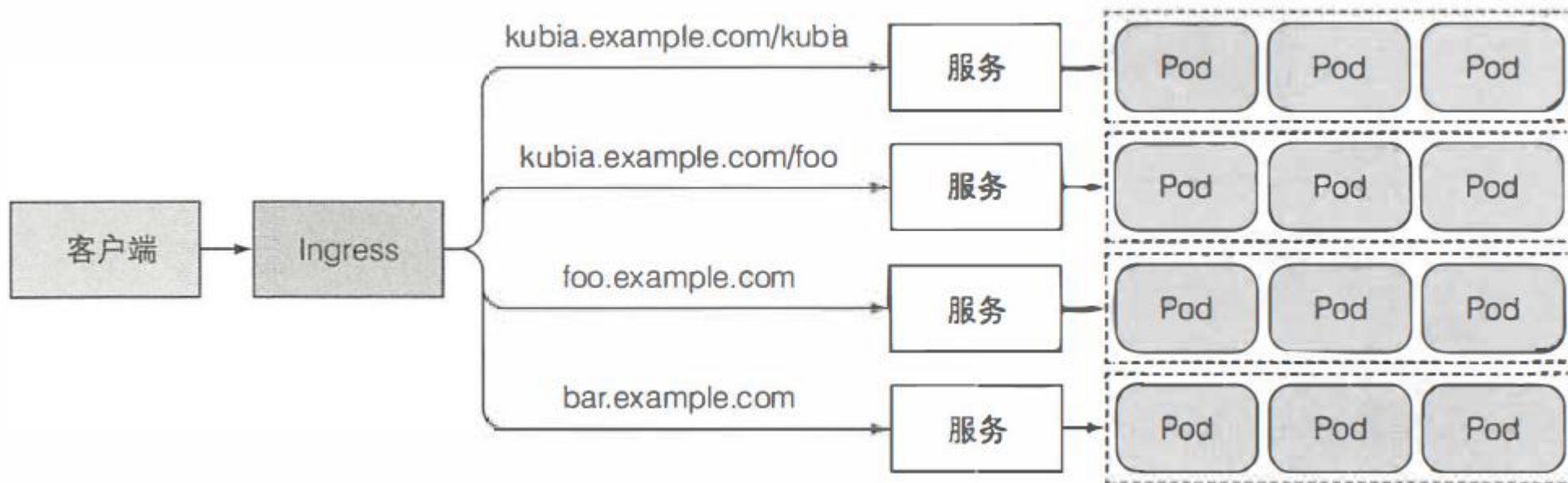
1. 实际提供服务的pod不在访问的node上
2. 在service模板中使用externalTrafficPolicy:local来将流量定向到本节点pod上:
 - 若本节点没有提供服务的pod则请求挂起
 - 若pod在节点中分布不均匀，则会造成每个pod负载不均
 - 设置该参数后，外部访问pod将不对源IP做SNAT转换

```
spec:  
  externalTrafficPolicy: Local  
  ...
```



4. Ingress ——介绍

1. nodeport、loadbalance方式需要端口、IP众多，ingress通过一个公共接口暴露多种服务
2. 通过主机名和路径决定使用的服务
3. 工作在第七层，可以提供cookie会话保持（使用域名访问）



4. Ingress ——介绍

Ingress使用反向代理负载均衡器来实现对外暴露服务，比如Nginx、Apache、Haproxy等。通过这种方式对外提供服务，一般包含3个组件：

1. 反向代理负载均衡器

负责拦截外部请求，读取Ingress定义的路由规则配置，转发相应的请求到后端服务。通常为nginx、apache、traefik等。

2. Ingress Controller

监听apiserver，实时感知Ingress路由规则集合的变化，获取service、pod等信息，然后发送给反向代理负载均衡器，刷新其路由配置信息，这就是它的服务发现机制。

3. Ingress

定义路由规则集合。

4. Ingress ——Ingress资源创建（集群已有控制器）

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


← Ingress 将域名 kubia.
example.com 映射到
你的服务

对 kubia.example.com/kubia 的请
求将会转发至 kubia 服务


对 kubia.example.com/bar 的请求
将会转发至 bar 服务

4. Ingress ——Ingress资源创建（集群已有控制器）

```
spec:
  rules:
  - host: foo.example.com
    http:
      paths:
      - path: /
        backend:
          serviceName: foo
          servicePort: 80
  - host: bar.example.com
    http:
      paths:
      - path: /
        backend:
          serviceName: bar
          servicePort: 80
```

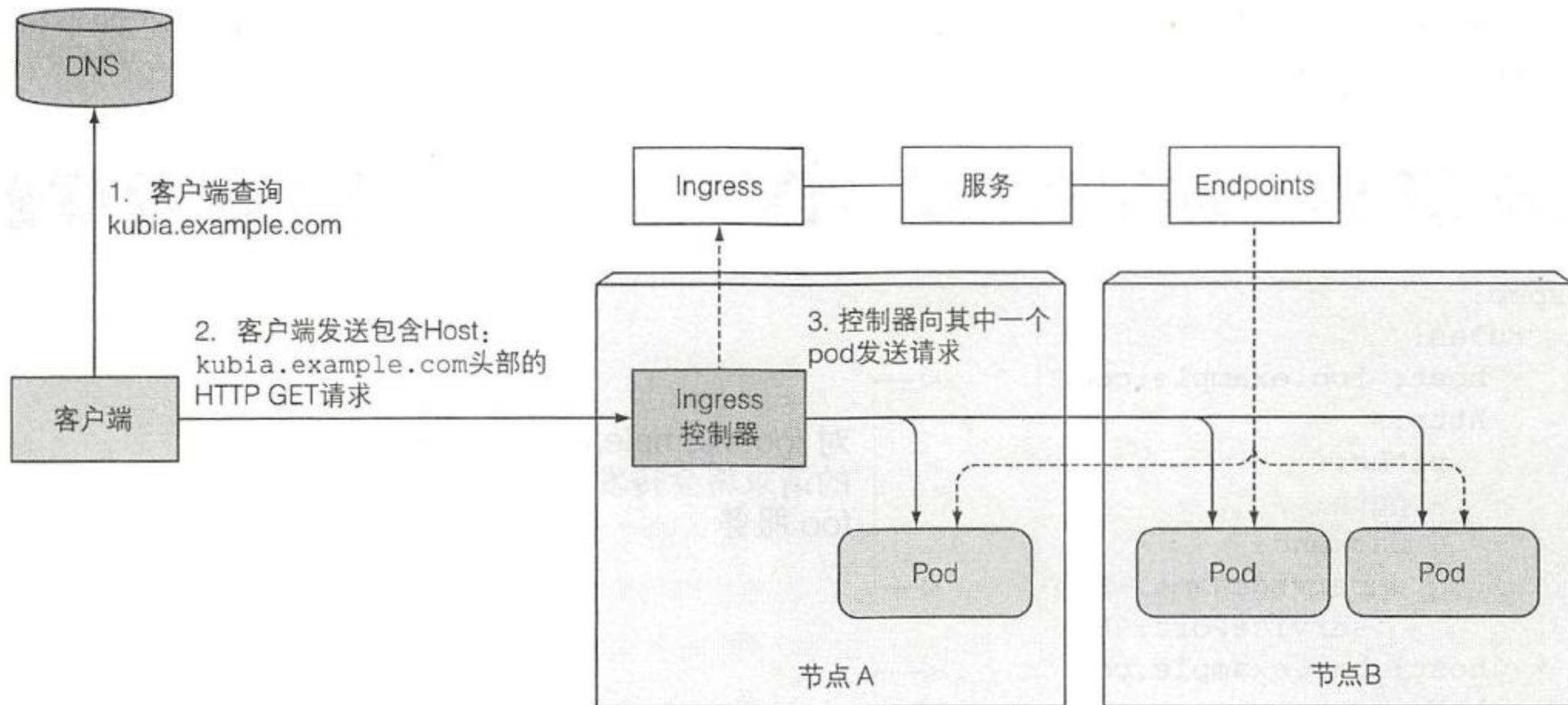


对 foo.example.com
的请求将会转发至
foo 服务



对 bar.example.com
的请求将会转发至
bar 服务

4. Ingress ——工作原理



4. Ingress ——配置TLS支持

Ingress使用TLS连接时：

- 客户端和控制器的连接是加密的，而控制器和后端pod的连接不是
- Ingress 控制器负责处理与TLS 相关的所有内容
- 需要创建自定义私钥和证书，然后使用私钥和证书创建tls类型的secret，并在创建ingress时指定该secret

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: kuba
spec:
  tls:
  - hosts:
    - kuba.example.com
    secretName: tls-secret
  rules:
  - host: kuba.example.com
    http:
      paths:
      - path: /
        backend:
          serviceName: kuba-nodeport
          servicePort: 80
```

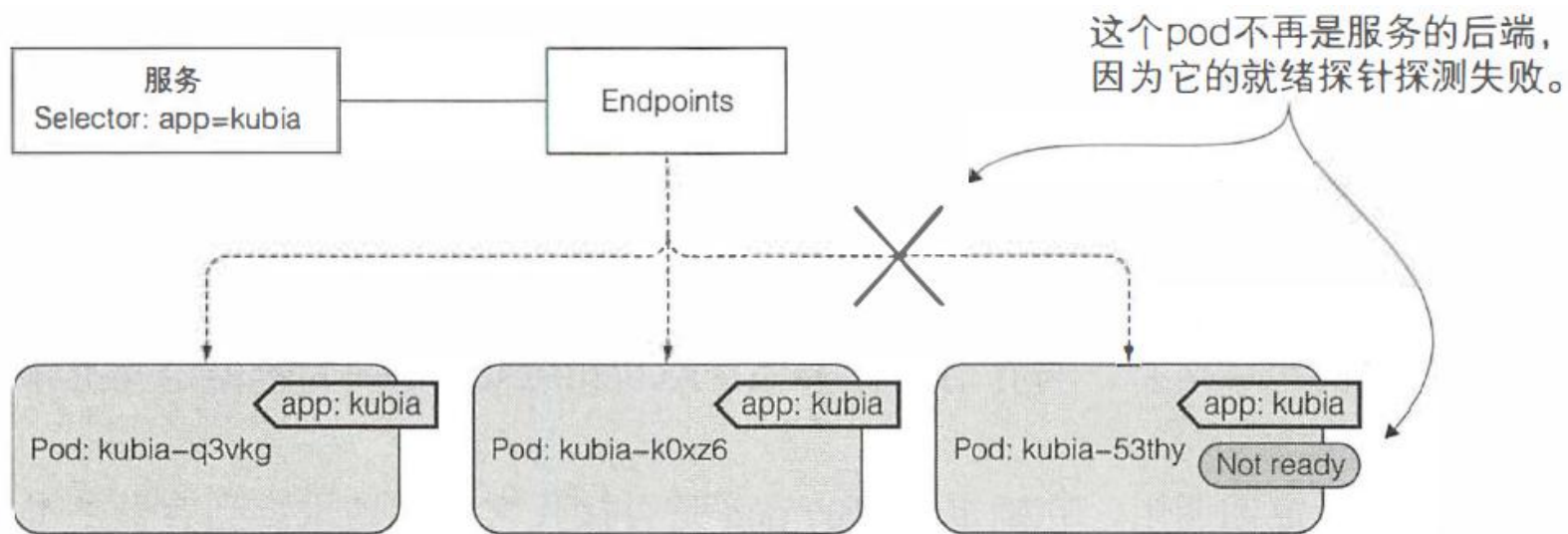
在这个属性下包含了所有的 TLS 的配置

将接收来自 kuba.example.com 主机的 TLS 连接

从 tls-secret 中获得之前创立的私钥和证书

5. Pod可用性判断 ——就绪探针

- ◆ 用于service、ingress判断pod是否已经就绪并可提供服务
- ◆ 容器启动后，经过等待时间后第一次执行，然后根据设置时间周期性调用
- ◆ 分为三种类型：
 - Exec探针：容器的状态由进程的退出状态代码确定
 - HTTP GET探针：向容器发送HTTP GET 请求，通过响应的HTTP 状态代码判断容器是否准备好
 - TCP socket探针：打开一个TCP 连接到容器的指定端口。如果连接已建立，则认为容器已准备就绪



5. Pod可用性判断 ——就绪探针

```
apiVersion: v1
kind: ReplicationController
...
spec:
  ...
  template:
    ...
    spec:
      containers:
      - name: kubia
        image: luksa/kubia
        readinessProbe:
          exec:
            command:
            - ls
            - /var/ready
        ...
```

pod 中的每个容器都会有一个就绪探针

5. Pod可用性判断 ——服务中发现未就绪的pod

```
kind: Service
metadata:
  annotations:
    service.alpha.kubernetes.io/tolerate-unready-endpoints: "true"
```

6. Headless服务

- ◆ Kubernetes 允许客户通过DNS 查找发现pod IP
- ◆ 当执行服务的DNS 查找时，DNS 服务器会返回单个IP——服务的集群IP

```
$ kubectl exec dnsutils nslookup kuba
...
Name:      kuba.default.svc.cluster.local
Address: 10.111.249.153
```

- ◆ 若在创建服务时将clusterIP设置为None，则返回的是服务下已就绪pod 的IP

```
apiVersion: v1
kind: Service
metadata:
  name: kuba-headless
spec:
  clusterIP: None
  ports:
    - port: 80
      targetPort: 8080
  selector:
    app: kuba
```

← 这使得服务成为 headless 的

```
$ kubectl exec dnsutils nslookup kuba-headless
...
Name:      kuba-headless.default.svc.cluster.local
Address: 10.108.1.4
Name:      kuba-headless.default.svc.cluster.local
Address: 10.108.2.5
```

感谢！