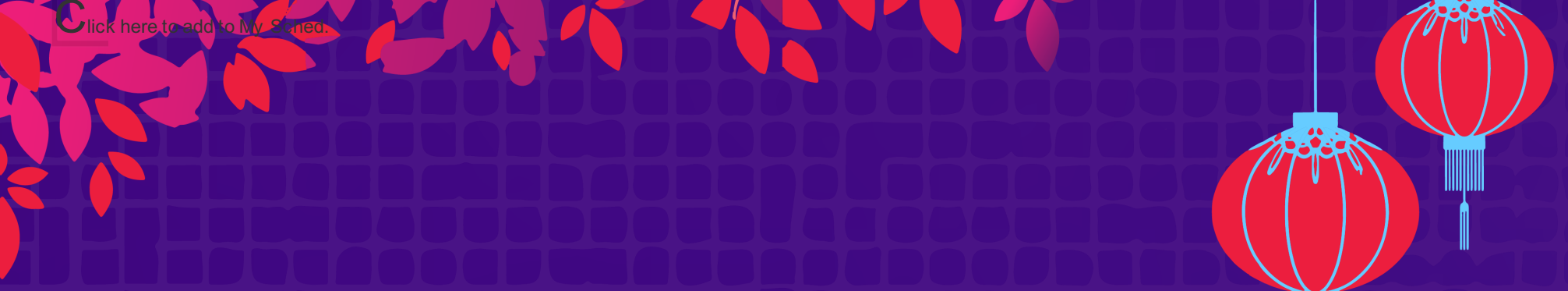

KubeCon
CloudNativeCon
OPEN SOURCE SUMMIT
China 2019



KubeCon | CloudNativeCon

OPEN SOURCE SUMMIT China 2019

Topology-aware Service Routing for Kubernetes is Coming Soon!

Jun Du

Who am I?

- Github: m1093782566
- CNCF TOC Contributor
- Kubernetes: Maintainer
- Istio: Approver
- KubeEdge: Maintainer
- K8s SIG Network：IPVS, CNI bandwidth, Topology-aware service routing
- 《云原生分布式存储基石：etcd》
- 《Docker容器与容器云》

Agenda

- Kubernetes亲和性系统设计
- 网络亲和性路由需求场景
- 网络亲和性路由设计方案
- Q&A

K8s三大亲核心设计

KubeCon CloudNativeCon OPEN SOURCE SUMMIT China 2019

Local Storage

(Pod <-> Volume)

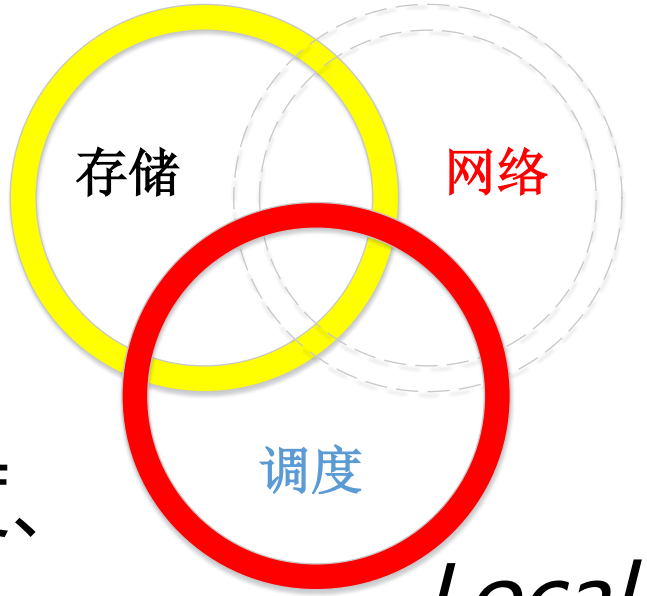
Topology-aware volume scheduling: v1.9

Topology-aware volume provision(static): v1.13

Local Endpoints

(Pod <-> Service backends)

Topology-aware service routing: coming soon!



三年磨一剑，补齐调度、存储、网络三大能力！

Local Node

(Pod <-> Node) Node Affinity: 1.2

(Pod <-> Pod) Pod Affinity: 1.4

如何表达亲和性？

- Host
- AZ
- Region
- Rack
- Generator
- SSD
- Anything you like…

Topology
拓扑域

- **亲和 = 在指定的拓扑域内**
- **反亲和 = 不在指定的拓扑域内**

- *如何表达拓扑域？*

  拓扑是任意的

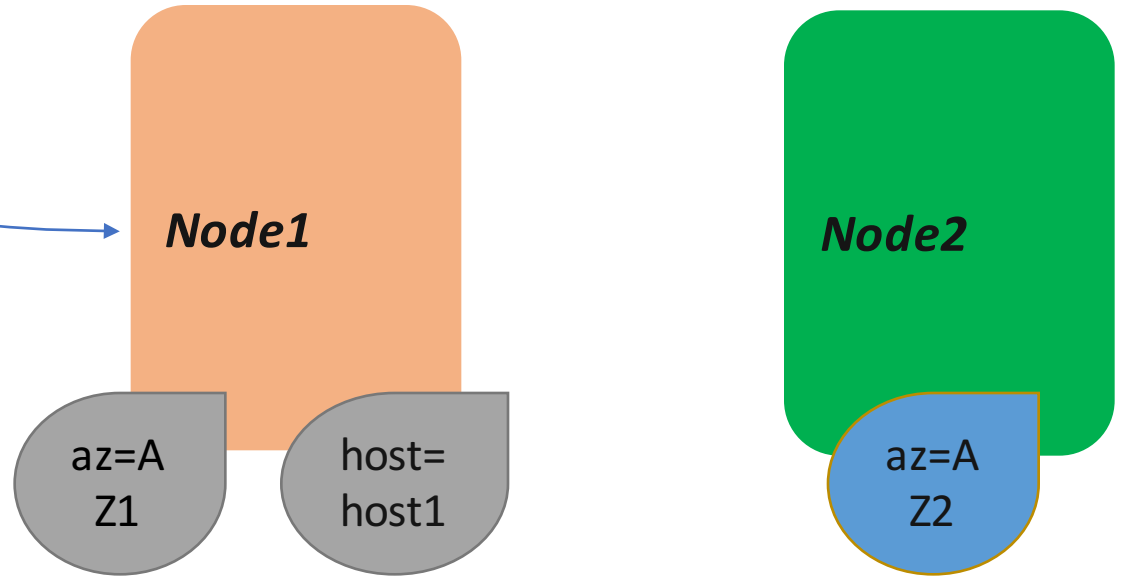
Topology-aware Scheduling in Kubernetes

Where should I run my applications?

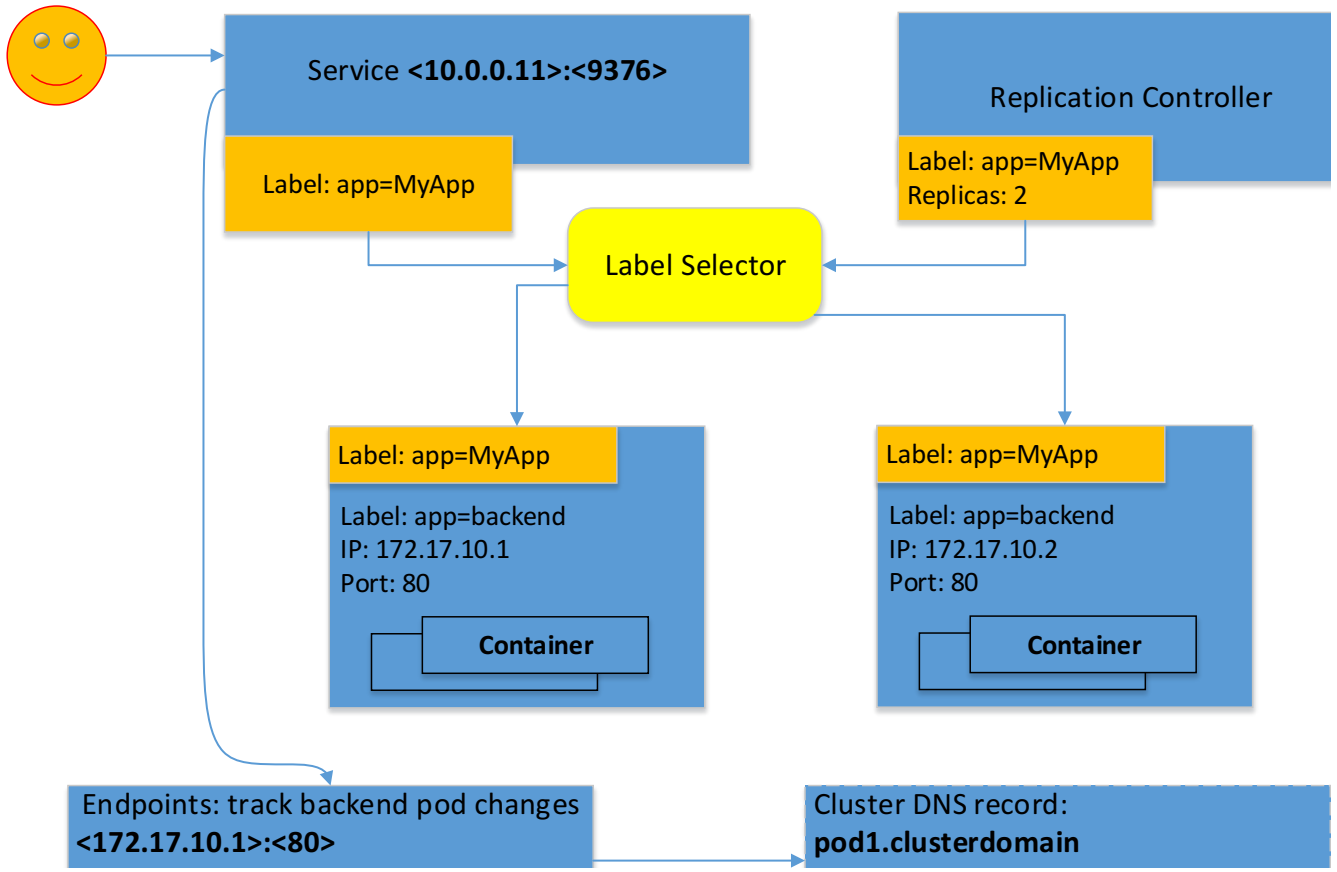
Scheduling is about finding hardware to run your code.

Node Affinity


```
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: az
            operator: In
            values:
            - az1
            - az2
      preferredDuringSchedulingIgnoredDuringExecution:
      - weight: 100
        preference:
          matchExpressions:
          - key: hostname
            operator: In
            values:
            - host1
```



K8s Service & Endpoints



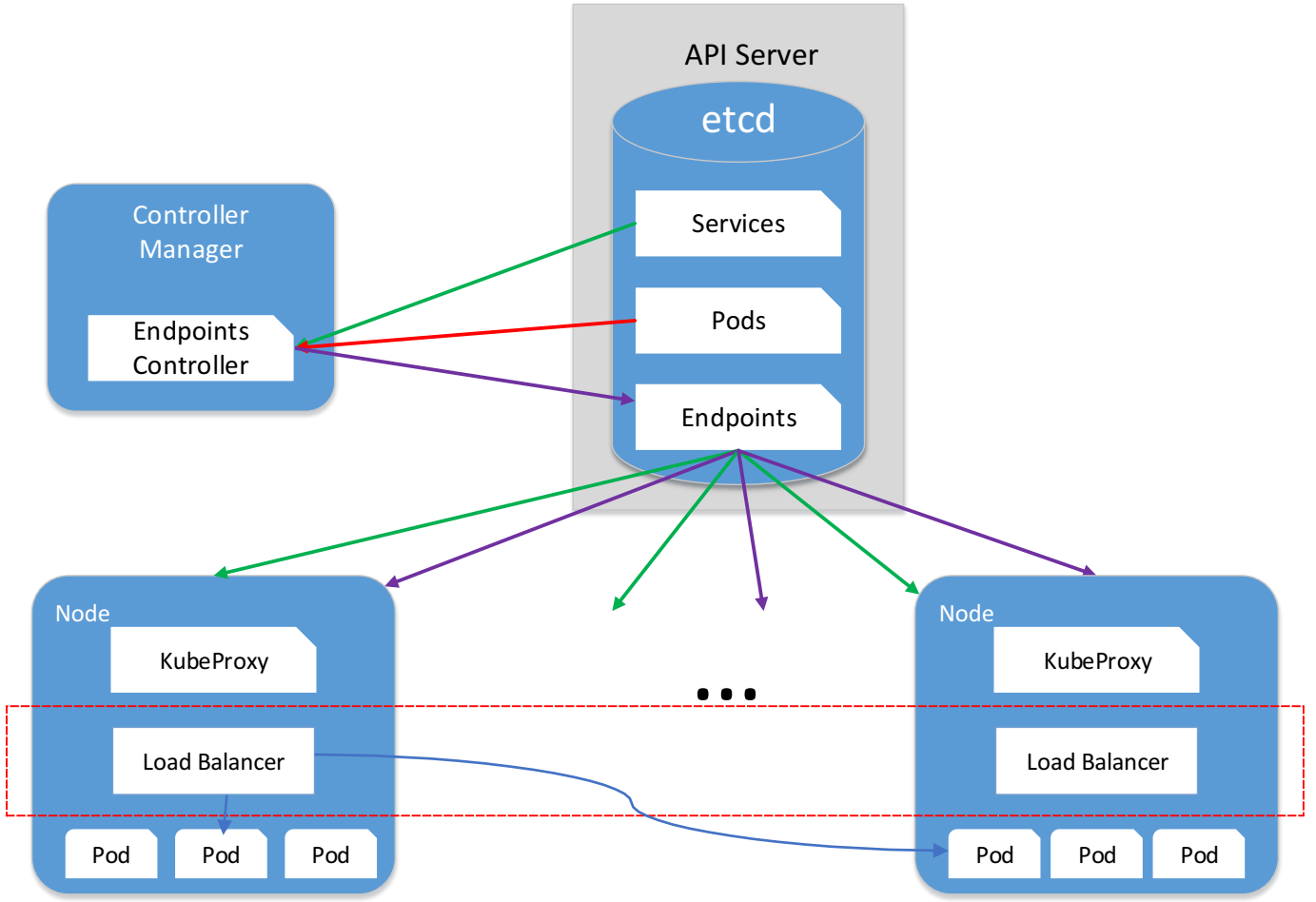
Service:

- A记录
  - 普通Service: svc.namespace.**svc.cluster.local→Cluster IP**
  - headless Service: vc.namespace.**svc.cluster.local→后端Pod IP列表**
- SRV记录：
  - _port-name._port-protocol.svc.namespace.svc.cluster.local→**Service Port**

Pod:

- A记录
  - **pod-ip**.namespace.pod.cluster.local → Pod IP

K8s Service内部机制



IPVS

iptables

Topology-aware Service Routing: 需求场景

- Host local的场景（ExternalTrafficPolicy）
  - daemonset部署的服务, .e.g. fluentd, aws-es-proxy
  - 安全
- Zone local的场景
  - 跨AZ流量收费
  - 性能

… 拓扑是任意的，用户的需求是无止境的…

用Labels描述拓扑域

Topology-aware Service Routing:问题澄清

- 硬亲和 VS. 软亲和？
  - 我就要那个！ ->> 你的小可爱不想跟你说话，并向你扔来一个connection refused!
  - 我能将就 ☺
- 如果是软亲和
  - 你有多爱我？ ->> 你给你的小可爱贴上了权重(weight)
- 如果有多个匹配的Endpoints
  - 按权重来？ ->> 幸运的是IPVS和iptables都支持wrr
  - 雨露均沾？
- 谁来配置？
  - 用户配置：参考 ExternalTrafficPolicy
  - 管理员配置：参考 Istio和NetworkPolicy

API设计

type ServiceSpec struct {

// topologyKeys is a ***preference-order list*** of ***topology keys*** which clients should respect when accessing this Service.

// If any ready backends exist for index [0], they should always be chosen; only if no backends exist for index [0] should backends for index [1+] be considered.

// If this field is specified, but all entries have no backends that match the topology of the client, the service has no backends for that client and connections should fail.

// ***The special value "" may be used to means "any node".*** This catch-all value, if used, only makes sense as the last value in the list.

// ***If this is not specified or empty, no topology contraints will be applied.***

// +optional

TopologyKeys []string

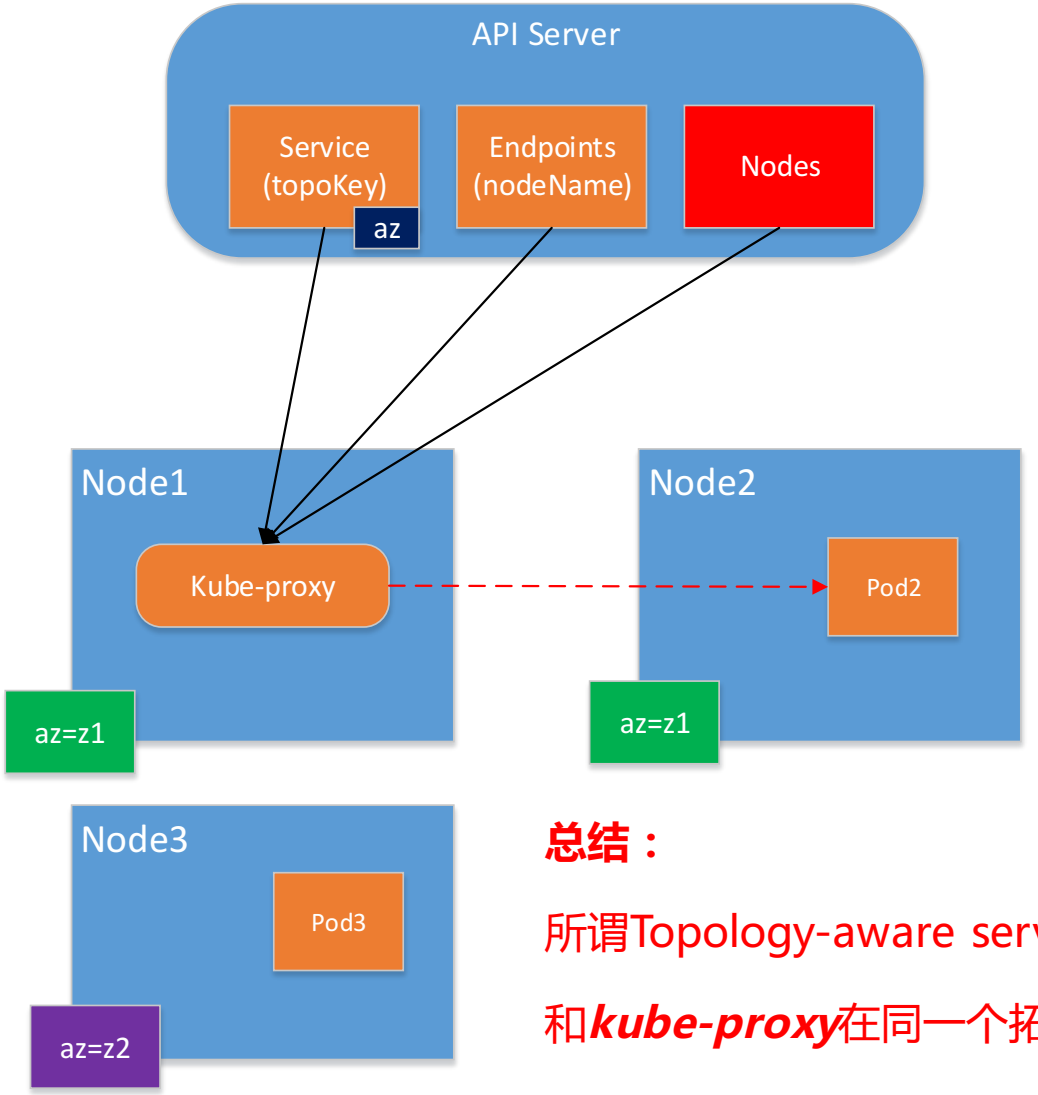
}

Kube-proxy新的工作流

- **Kube-proxy会watch所有Nodes！（*对，你没看错*）**

watch的开销由数据变化频率决定，而不是数据量！！

- Node的拓扑域由Node Labels表达
- Pod的拓扑域由所在Node的拓扑域决定
- Kube-proxy已经get了自己的Node Name，再去查本地watch的Node缓存，也就知道了自己所在的拓扑域
- Kube-proxy通过Endpoints的nodeName知道了其对应Pod所在节点的Node Name，再去查Nodes（Node Labels），知道了和他在同一个拓扑域的所有Endpoints



总结：

所谓Topology-aware service routing就是返回和***kube-proxy***在同一个拓扑域的后端Pod！

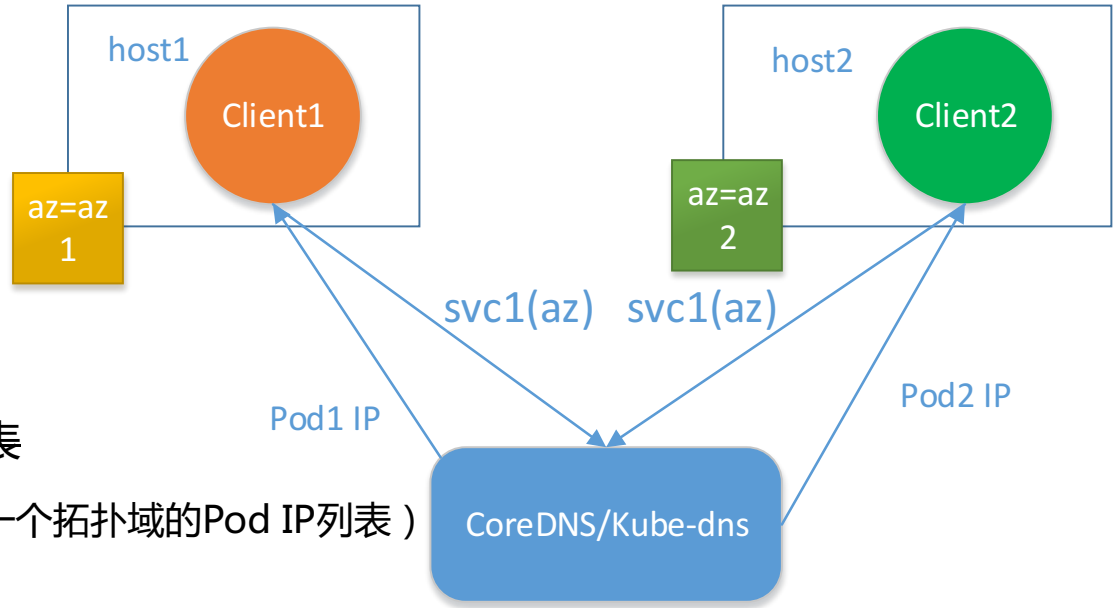
等等！不要忘了headless service

- DNS Server:

  - watch Service & Endpoints

  - svc.namespace.svc.cluster.local → ~~后端Pod IP列表~~

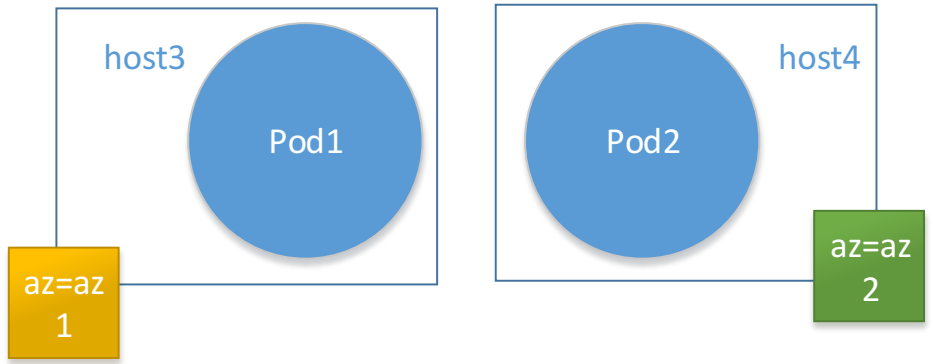
    （和***客户端***在同一个拓扑域的Pod IP列表）



问题：如何获得Client Pod所在的拓扑域？

~~Watch所有Pod和Node？~~

需要找到Pod IP -> Node Labels的映射关系



PodLocator CRD


```
// PodLocator represents information about where a pod exists in arbitrary space.  This is useful for things like
// being able to reverse-map pod IPs to topology labels, without needing to watch all Pods or all Nodes.
type PodLocator struct {
  metav1.TypeMeta
  // +optional
  metav1.ObjectMeta
  // IP addresses of the running Pod.
  // May not be loopback (127.0.0.0/8), link-local (169.254.0.0/16),
  // or link-local multicast ((224.0.0.0/24).
  // Both IPv4 and IPv6 are also accepted.
  // +optional
  addresses []string
  // Name of Node where the Pod is running on.
  // +optional
  NodeName string
  // Labels of Node where the Pod is running on, being added for identifying Pod's topological information.
  // +optional
  NodeLabels map[string]string
}
```

幸运的是，beta之前我们不要考虑实现这个CRD ☺

Thanks!