法律声明

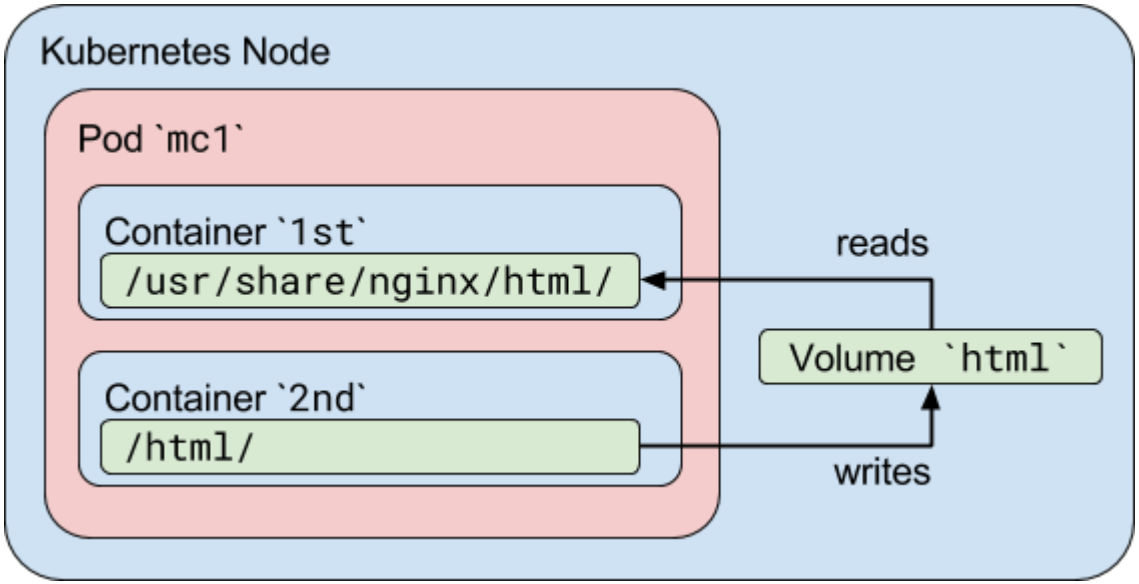
- 本课件包括：演示文稿，示例，代码，题库，视频和声音等，讲师及小象学院拥有完全知识产权的权利；只限于善意学习者在本课程使用，不得在课程范围外向任何第三方散播。任何其他人或机构不得盗版、复制、仿造其中的创意，我们将保留一切通过法律手段追究违反者的权利。

- 课程详情请咨询
  - 微信公众号：小象
  - 新浪微博：ChinaHadoop

Kubernetes应用管理-上

目录

1. Pod详解



- Pod就像是豌豆荚一样，它由一个或者多个容器组成
- Pod中的容器共享IP地址和端口号，它们之间可以通过localhost互相发现。它们之间可以通过进程间通信，例如SystemV信号或者POSIX共享内存。不同Pod之间的容器具有不同的IP地址，不能直接通过IPC通信。
- Pod中的容器也有访问共享volume的权限，这些volume会被定义成pod的一部分并挂载到应用容器的文件系统中。

1. Pod的设计动机

- 容器介于操作系统和应用之间，容器的推荐玩法是每个容器运行一个进程。
  - 外部掌控多容器的组合和生命周期 — redhat和docker公司的控制权之争
  - 单容器多进程玩法的案例 — 阿里的pouch项目
- 对外，Pod作为一个独立的部署单位，支持横向扩展和复制。共生（协同调度），命运共同体（例如被终结），协同复制，资源共享，依赖管理
- 对内，Pod内容器互相协作
  - pod中的应用必须协调端口占用。每个pod都有一个唯一的IP地址，跟物理机和其他pod都处于一个扁平的网络空间中，它们之间可以直接连通。
  - Pod中应用容器的hostname被设置成Pod的名字。
  - Pod中的应用容器可以共享volume。Volume能够保证pod重启时使用的数据不丢失。

1.Pod的非持久性

- Pod在以下几种情况下都会死
  - 调度失败
  - 节点故障
  - 缺少资源
  - 节点维护
  - 用户主动干掉Pod
- 问题：为什么Pod要被设计成这么容易死？

1. Init 容器

- Init 容器是一种专用的容器，在应用程序容器启动之前运行
- Init 容器总是运行到成功完成为止。
- 每个 Init 容器都必须在下一个 Init 容器启动之前成功完成。
- Init 容器能做什么？
  - 等待一个 Service 创建完成，通过类似如下 shell 命令：

for i in {1..100}; do sleep 1; if dig myservice; then exit 0; fi; exit 1

  - 在启动应用容器之前等一段时间，使用类似 sleep 60 的命令。

1. Pod中容器镜像 & 其他

pod.spec.containers[]. imagePullPolicy	下载策略
IfNotPresent(默认)	在没有此镜像时下载
Always	每次必下载

镜像的tag设置	下载策略
image:lastest	每次必下载
image	每次必下载
image:*	在没有此镜像时下载

扩展阅读

1. 私有镜像仓库的下载
2. 环境变量
3. 生命周期hook

1. Pod.status.phase

挂起（Pending）	Pod 已被 Kubernetes 系统接受，但有一个或者多个容器镜像尚未创建。等待时间包括调度 Pod 的时间和通过网络下载镜像的时间，这可能需要花点时间。
运行中（Running）	该 Pod 已经绑定到了一个节点上，Pod 中所有的容器都已被创建。至少有一个容器正在运行，或者正处于启动或重启状态。
成功（Successed）	Pod 中的所有容器都被成功终止，并且不会再重启。
失败（Failed）	Pod 中的所有容器都已终止了，并且至少有一个容器是因为失败终止。也就是说，容器以非0状态退出或者被系统终止。
未知（Unkonwn）	因为某些原因无法取得 Pod 的状态，通常是因为与 Pod 所在主机通信失败。

- Pod.status.condition…

1. 容器探针(注意不是Pod探针)

- 探针探查方式
  - ExecAction：在容器内执行指定命令。如果命令退出时返回码为 0 则认为诊断成功。
  - TCPSocketAction：对指定端口上的容器的 IP 地址进行 TCP 检查。如果端口打开，则诊断被认为是成功的。
  - HTTPGetAction：对指定的端口和路径上的容器的 IP 地址执行 HTTP Get 请求。如果响应的状态码大于等于200 且小于 400，则诊断被认为是成功的。
- 探针结果：成功、失败、未知
- 探针引起外部动作
  - livenessProbe：指示容器是否正在运行。如果存活探测失败，则 kubelet 会杀死容器，并且容器将受到其 重启策略 的影响。如果容器不提供存活探针，则默认状态为 Success。
  - readinessProbe：指示容器是否准备好服务请求。如果就绪探测失败，端点控制器将从与 Pod 匹配的所有 Service 的端点中删除该 Pod 的 IP 地址。初始延迟之前的就绪状态默认为 Failure。如果容器不提供就绪探针，则默认状态为 Success

1. 容器重启策略

Pod.spec.restartPolicy字段	行为：适用于Pod中所有失败的容器
Always（默认）	exitCode=任何数字，执行重启操作
OnFailure	exitCode!=0，执行重启操作
Never	exitCode=任何数字，不重启

以五分钟为上限的指数退避延迟（10秒，20秒，40秒...）重新启动，并在成功执行十分钟后重置。

文档地址： Kubectl run -h

1. Pod状态 = 容器状态之和?

容器	退出状态	log event	restartPolicy	行为	Pod.status.phase
1个	success	completion	Always	重启容器	Running
			OnFailure	无	Succeeded
			Never	无	Succeeded
1个	fail	failure	Always	重启容器	Running
			OnFailure	重启容器	Running
			Never	无	Failed
1 2	fail 正常运行	failure	Always	重启容器	Running
			OnFailure	重启容器	Running
			Never	无	Running
1 2	not running exit	failure	Always	重启容器	Running
			OnFailure	重启容器	Running
			Never	无	Failed

1. Pod状态 = 容器状态之和?

容器	退出状态	log event	restartPolicy	行为	Pod.status.phase
1个	out of memeory	OOM	Always	重启容器	Running
			OnFailure	重启容器	Running
			Never	无	Failed
n个	磁盘挂了	failure	A/O/N	杀死所有容器，如果pod被管控，异地重建	Failed
n个	node被分区	failure	A/O/N	等待node超时，如果pod被管控，异地重建	Failed

2. ReplicaSet： 副本集

```
apiVersion: extensions/v1beta1
kind: ReplicaSet
metadata:
  name: frontend
spec:
replicas: 3
selector:
    matchLabels:
      tier: frontend
    matchExpressions:
      - {key: tier, operator: In, values: [frontend]}
```

1. 自动插入pod template里面的label：
 # labels:
   # app: guestbook
   # tier: frontend

2. 维持3副本，可被修改

3. 选择器，也是自动从pod template里面获取，这里演示下表达式

2. ReplicaSet： 副本集

```
template:
  metadata:
    labels:
      app: guestbook
      tier: frontend
  spec:
    containers:
    - name: php-redis
      image: gcr.io/google_samples/gb-frontend:v3
      resources:
        requests:
          cpu: 100m
          memory: 100Mi
      ports:
      - containerPort: 80
```

1. pod template，作为后续创建所有Pod的模板

2. 资源限制，详见此文《kubernetes中容器资源控制的那些事儿》

2. Deployment的行为和定义

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 3
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80
```

- 定义Deployment来创建Pod和ReplicaSet
- 滚动升级和回滚应用
- 扩容和缩容
- 暂停和继续Deployment

实践题：
Deployment和ReplicaSet有什么区别？

2. Deployment操作

扩容：

kubectl scale deployment nginx-deployment --replicas 10

如果集群支持 horizontal pod autoscaling 的话，

还可以为Deployment设置自动扩展：

kubectl autoscale deployment nginx-deployment --min=10 --max=15 --cpu-percent=80

更新镜像也比较简单:

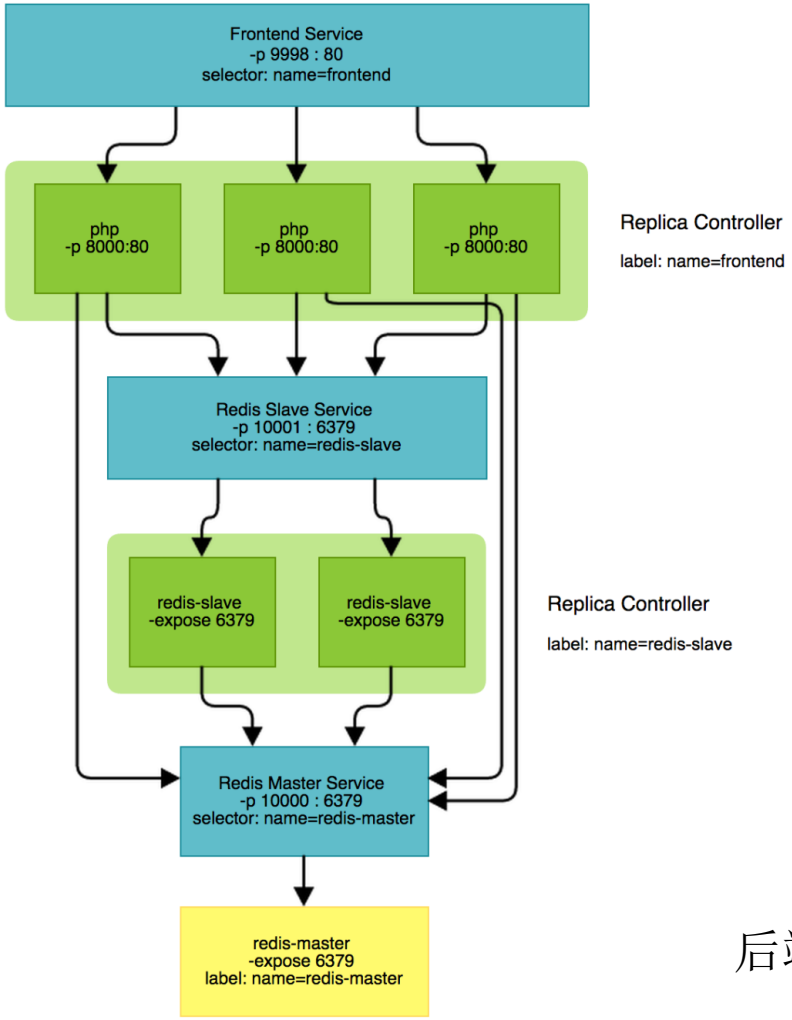
kubectl set image deployment/nginx-deployment nginx=nginx:1.9.1

回滚：

kubectl rollout undo deployment/nginx-deployment

《Deployment详解》

3. Services实操




```
kind: Service
apiVersion: v1
metadata:
  name: my-service
spec:
  selector:
    app: MyApp
  ports:
    - protocol: TCP
      port: 80
      targetPort: 9376
```

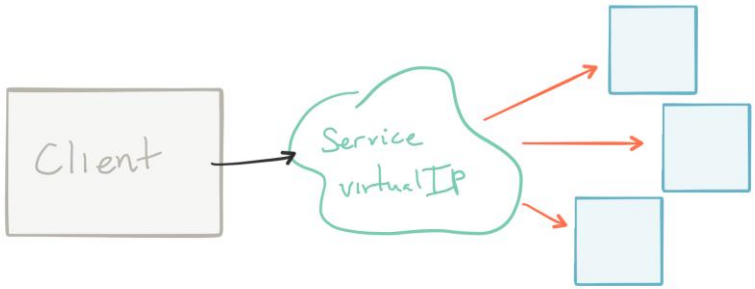
后端匹配→代理→发布→服务发现→客户端

3. Services → Pod → Endpoints

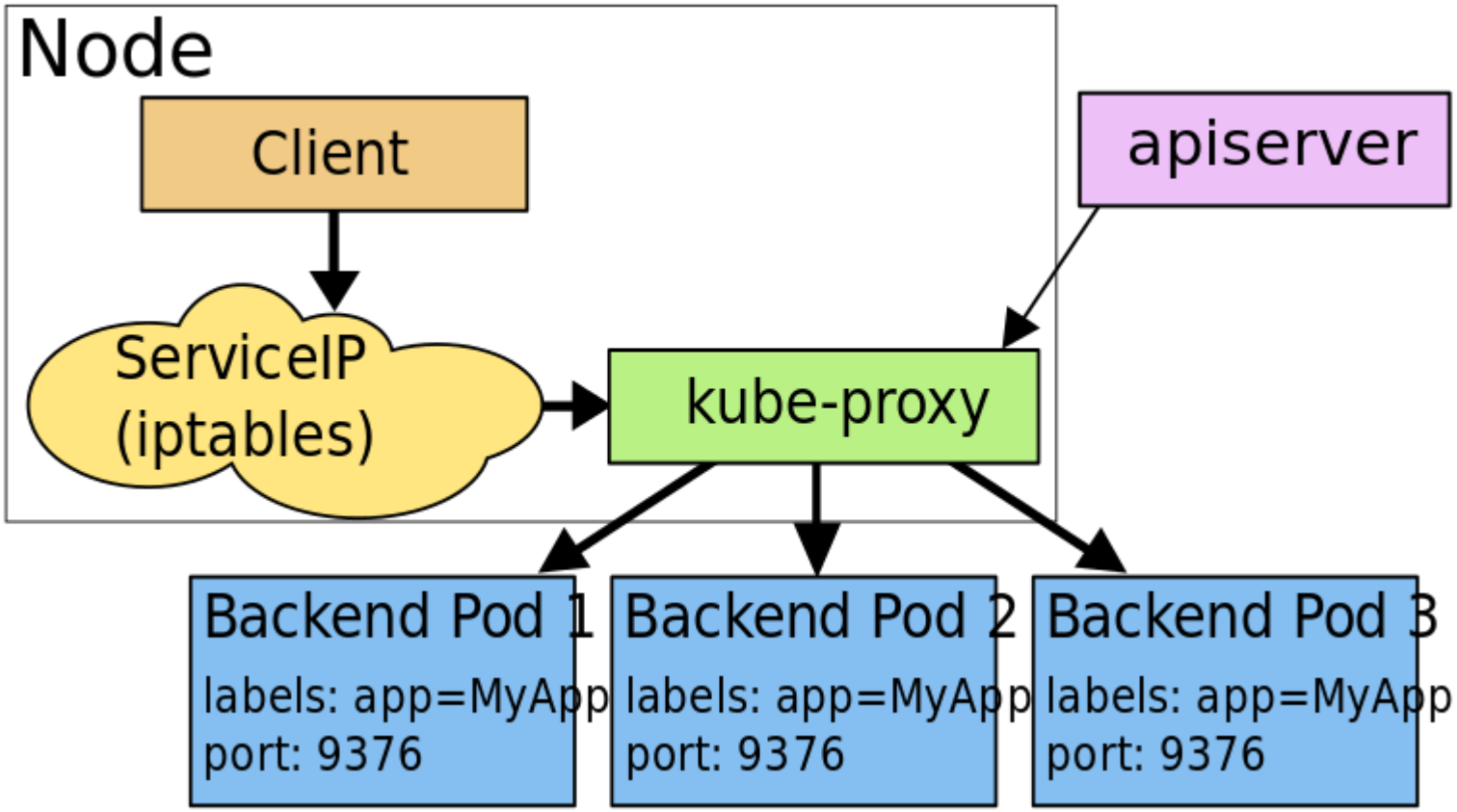
```
kind: Service
apiVersion: v1
metadata:
  name: my-service
spec:
  selector:
    app: MyApp
  ports:
    - protocol: TCP
      port: 80
      targetPort: 9376
```

```
kind: Pod
metadata:
  name: nginx-deployment
  labels:
      app: MyApp
spec:
    containers:
      ports:
      - containerPort: 9376
```

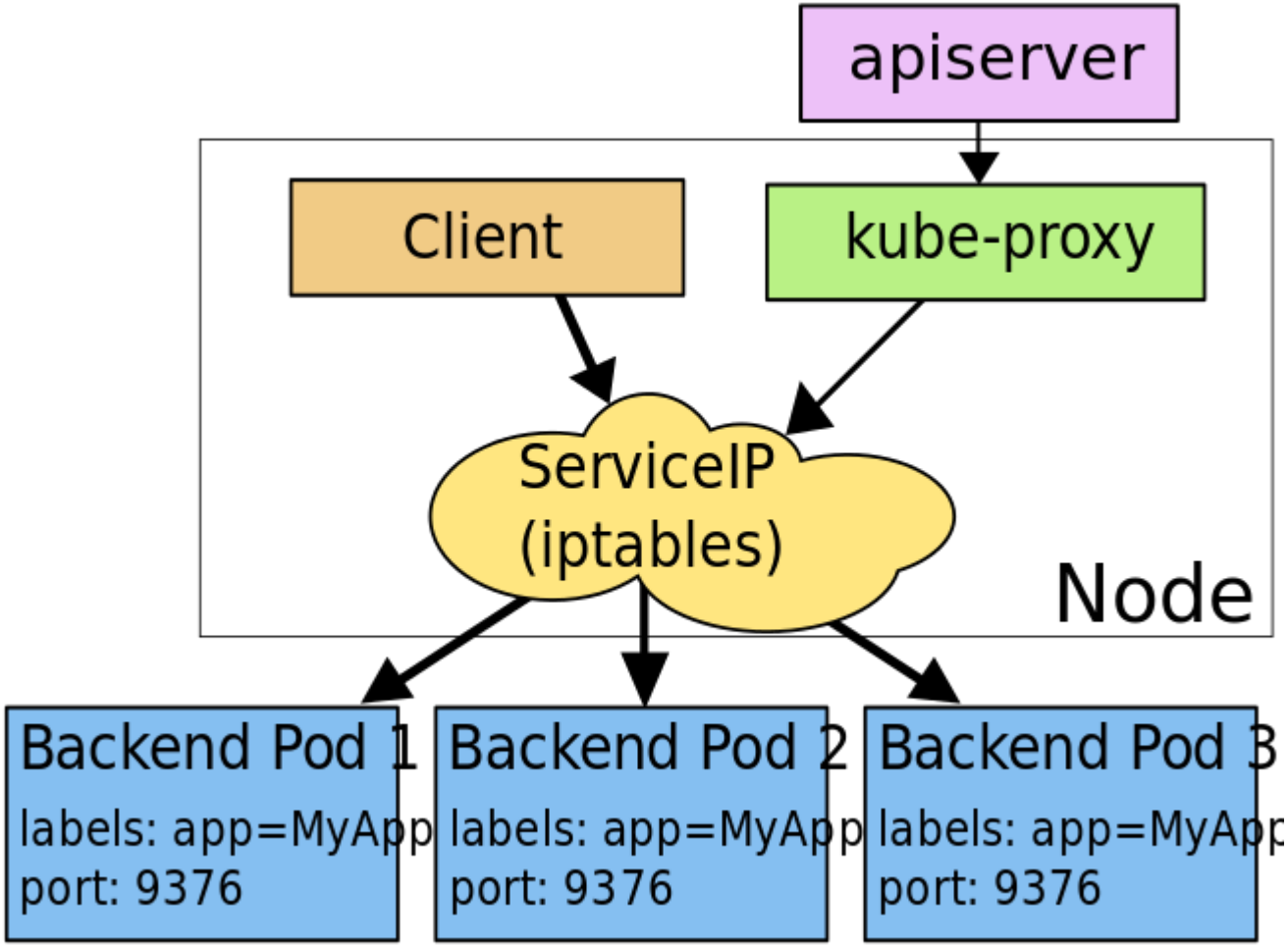
```
kind: Endpoints
apiVersion: v1
metadata:
  name: my-service
subsets:
  - addresses:
      - ip: 1.2.3.4
    ports:
      - port: 9376
```



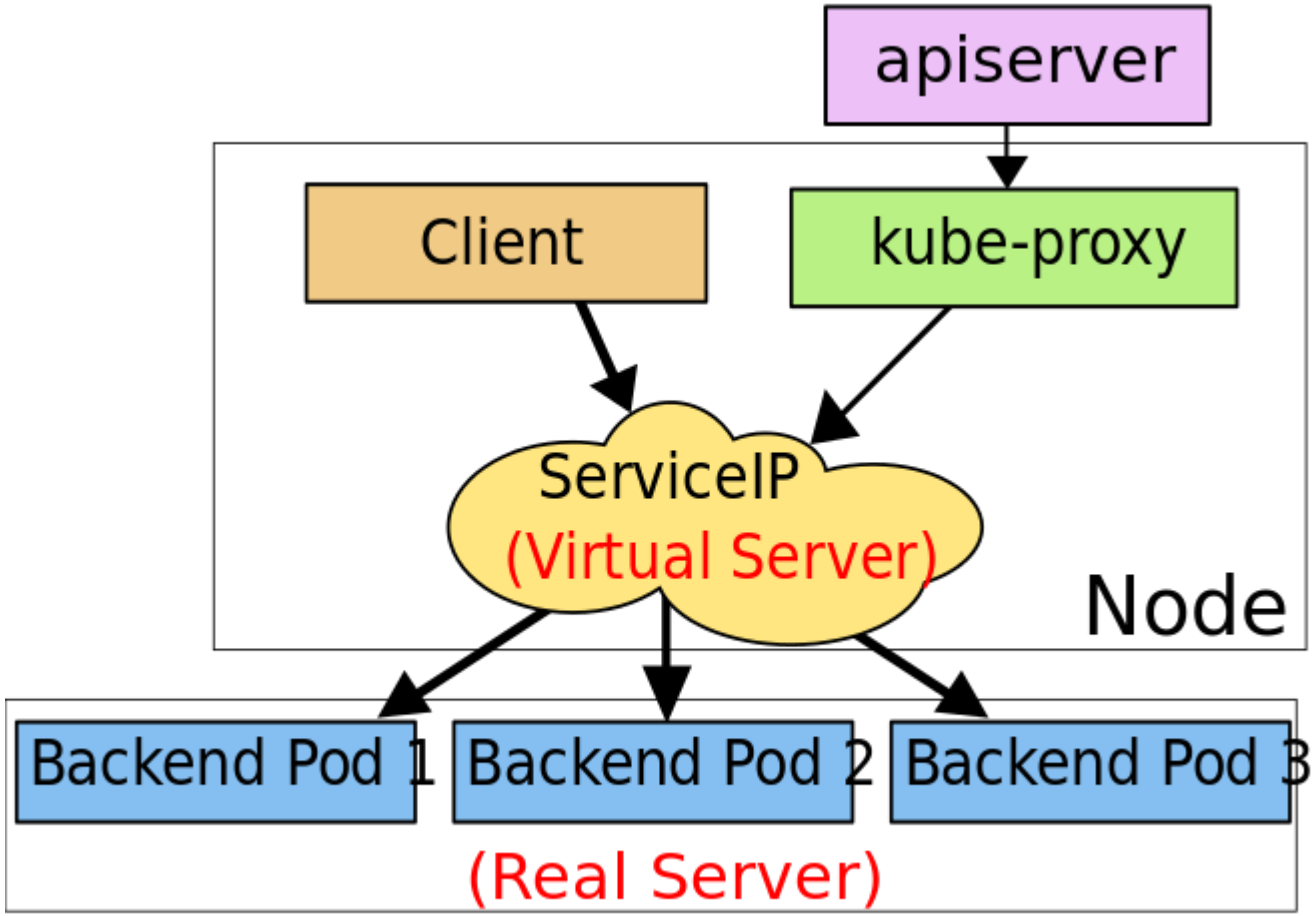
3. VIP：kube-proxy： Proxy-mode: userspace



3. VIP： kube-proxy： Proxy-mode: iptables



3. VIP： kube-proxy： Proxy-mode: ipvs(1.9 beta)



3. Service的发布类型

```
kind: Service
apiVersion: v1
metadata:
  name: my-service
spec:
  selector:
    app: MyApp
  ports:
  - protocol: TCP
    port: 80
    targetPort: 9376
  clusterIP: 10.0.171.239
  loadBalancerIP: 78.11.24.19
  type: LoadBalancer
status:
  loadBalancer:
    ingress:
    - ip: 146.148.47.155
```

type	说明
ClusterIP	通过集群的内部 IP 暴露服务，选择该值，服务只能够在集群内部可以访问，这也是默认的 ServiceType
NodePort	通过每个 Node 上的 IP 和静态端口（NodePort）暴露服务。NodePort 服务会路由到 ClusterIP 服务，这个 ClusterIP 服务会自动创建。通过请求 <NodeIP>:<NodePort>，可以从集群的外部访问一个 NodePort 服务
LoadBalancer	使用云提供商的负载均衡器，可以向外部暴露服务。外部的负载均衡器可以路由到 NodePort 服务和 ClusterIP 服务。
ExternalName	通过返回 CNAME 和它的值，可以将服务映射到 externalName 字段的内容（例如， foo.bar.example.com）。 没有任何类型代理被创建，这只有 Kubernetes 1.7 或更高版本的 kube-dns 才支持。

3. Service的发布类型+

```
kind: Service
apiVersion: v1
metadata:
  name: my-service
spec:
  selector:
    app: MyApp
  ports:
  - protocol: TCP
    port: 80
    targetPort: 9376
  externalIPs: - 80.11.12.10
```

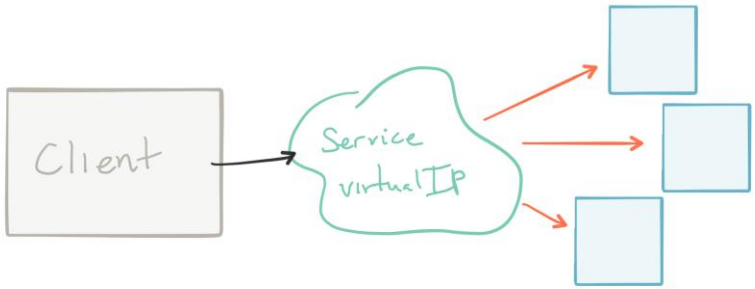
type	说明
externalIPs	如果外部的 IP 路由到集群中一个或多个 Node 上，Kubernetes Service 会被暴露给这些 externalIPs。通过外部 IP（作为目的 IP 地址）进入到集群，打到 Service 的端口上的流量，将会被路由到 Service 的 Endpoint 上。externalIPs 不会被 Kubernetes 管理，它属于集群管理员的职责范畴。

3. Services → 无Selector

```
kind: Service
apiVersion: v1
metadata:
  name: my-service
spec:
  selector:
    app: MyApp
  ports:
    - protocol: TCP
      port: 80
      targetPort: 9376
```

```
kind: Endpoints
apiVersion: v1
metadata:
  name: my-service
subsets:
  - addresses:
      - ip: 1.2.3.4
    ports:
      - port: 9376
```

手写

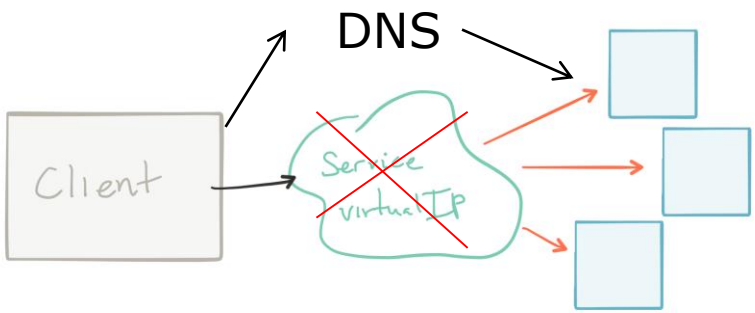


3. Services → Headless Services

```
kind: Service
apiVersion: v1
metadata:
  name: my-service
spec:
   clusterIP: None
  selector:
    app: MyApp
  ports:
    - protocol: TCP
      port: 80
      targetPort: 9376
```

```
kind: Pod
metadata:
  name: nginx-
deployment
  labels:
        app: MyApp
spec:
      containers:
       ports:
        - containerPort: 9376
```

```
kind: Endpoints
apiVersion: v1
metadata:
  name: my-service
subsets:
  - addresses:
      - ip: 1.2.3.4
    ports:
      - port: 9376
```

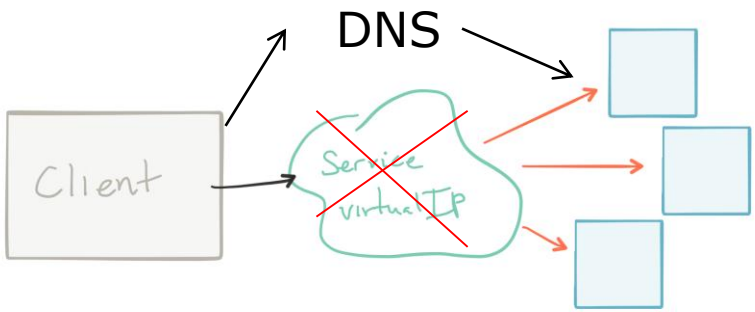


3. Services → Headless Services → 无Selector

```
kind: Service
apiVersion: v1
metadata:
  name: my-service
spec:
   clusterIP: None
  selector:
   app: MyApp
  ports:
    - protocol: TCP
      port: 80
      targetPort: 9376
```

```
kind: Endpoints
apiVersion: v1
metadata:
  name: my-service
subsets:
  - addresses:
      - ip: 1.2.3.4
    ports:
      - port: 9376
```

手写



3. 服务发现：环境变量

- 举个例子，一个名称为 "redis-master" 的 Service 暴露了 TCP 端口 6379，同时给它分配了 Cluster IP 地址 10.0.0.11，这个 Service 生成了如下环境变量：

- 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

- 约束：先有Service，后有Pod

3. 服务发现：DNS

- 一个可选（尽管强烈推荐）集群插件 是 DNS 服务器。 DNS 服务器监视着创建新 Service 的 Kubernetes API，从而为每一个 Service 创建一组 DNS 记录。 如果整个集群的 DNS 一直被启用，那么所有的 Pod应该能够自动对 Service 进行名称解析。

- 例如，有一个名称为 "my-service" 的 Service，它在 Kubernetes 集群中名为 "my-ns" 的 Namespace中，为 "my-service.my-ns" 创建了一条 DNS 记录。 在名称为 "my-ns" 的 Namespace 中的 Pod 应该能够简单地通过名称查询找到 "my-service"。 在另一个 Namespace 中的 Pod 必须限定名称为 "my-service.my-ns"。 这些名称查询的结果是 Cluster IP。

- Kubernetes 也支持对端口名称的 DNS SRV（Service）记录。 如果名称为 "my-service.my-ns" 的 Service 有一个名为 "http" 的 TCP 端口，可以对 "_http._tcp.my-service.my-ns" 执行 DNS SRV 查询，得到 "http" 的端口号。

3. 客户端访问

访问方式	集群内部Pod	集群外部
环境变量	O（有顺序约束）	X
DNS	O	X（除非级联）
externalIPs	O（看网络配置）	O
ClusterIP	O	X
NodePort	O	O
LoadBalancer	X	O

4. DaemonSet

DaemonSet 确保全部（或者一些）Node 上运行一个 Pod 的副本。当有 Node 加入集群时，也会为他们新增一个 Pod 。当有 Node 从集群移除时，这些 Pod 也会被回收。删除 DaemonSet 将会删除它创建的所有 Pod。

使用 DaemonSet 的一些典型用法：

- 运行集群存储 daemon，例如在每个 Node 上运行 glusterd、ceph。
- 在每个 Node 上运行日志收集 daemon，例如fluentd、logstash。
- 在每个 Node 上运行监控 daemon，例如 Prometheus Node Exporter、collectd、Datadog 代理、New Relic 代理，或 Ganglia gmond。

DaemonSet	ReplicaSet/Deployment
默认为每个Node上运行	不绑定Node，只保持指定个数
>1.6后自身支持滚动升级	通过Deployment支持滚动升级

作业

- 使用kubectl把上面的各种应用类型都玩一遍

联系我们

小象学院：互联网新技术在线教育领航者

- 微信公众号：大数据分析挖掘
- 新浪微博：ChinaHadoop