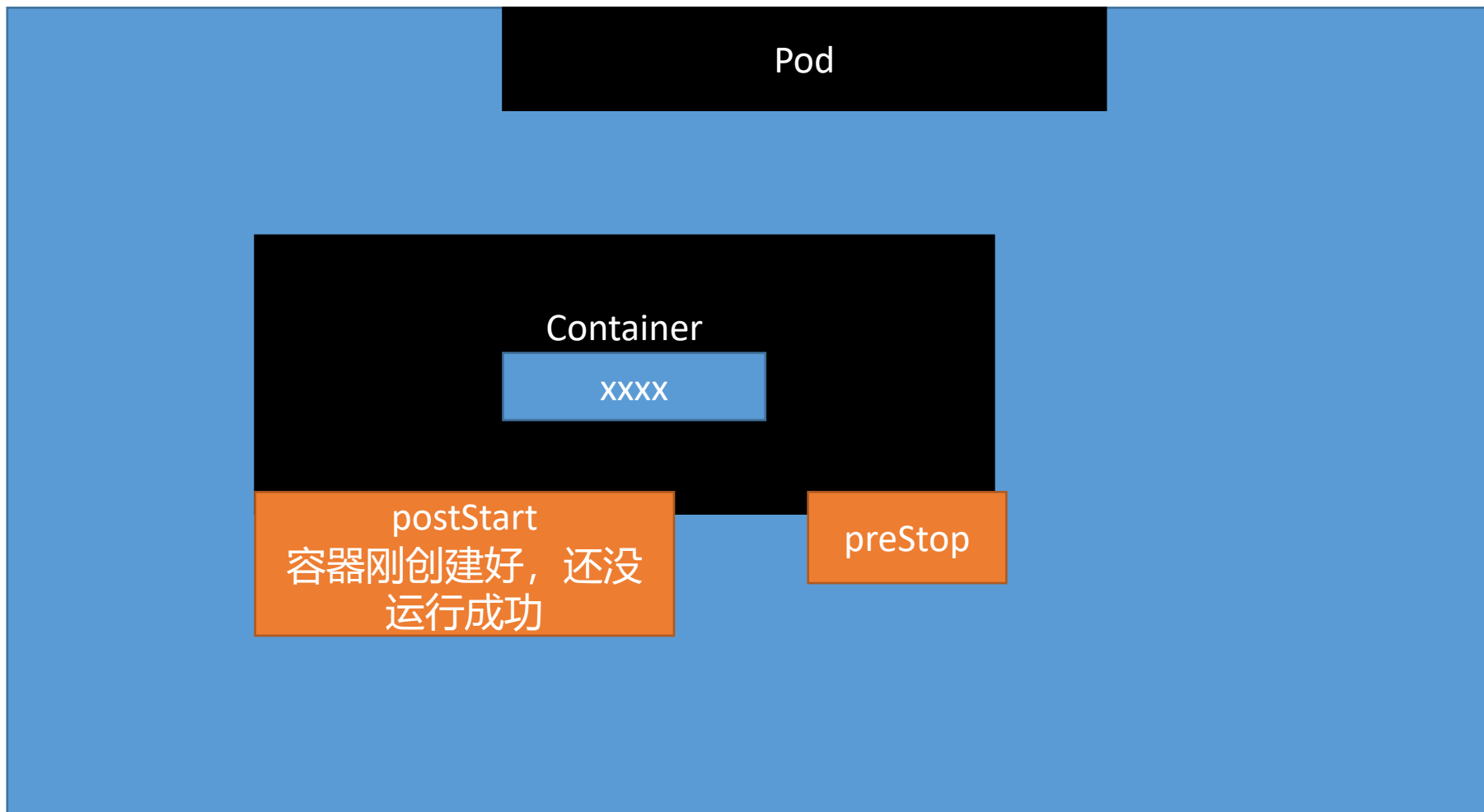
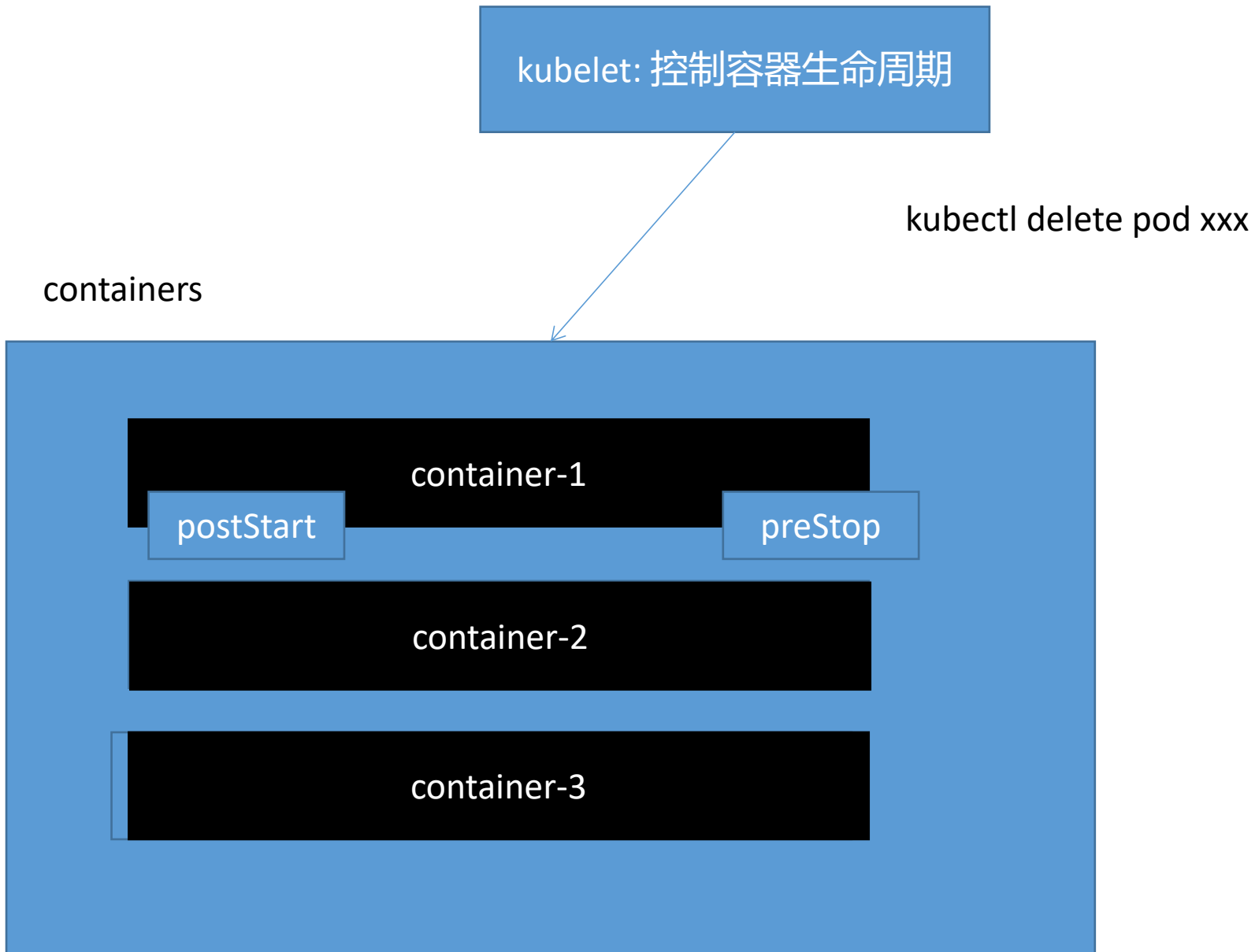


K8S图例

容器



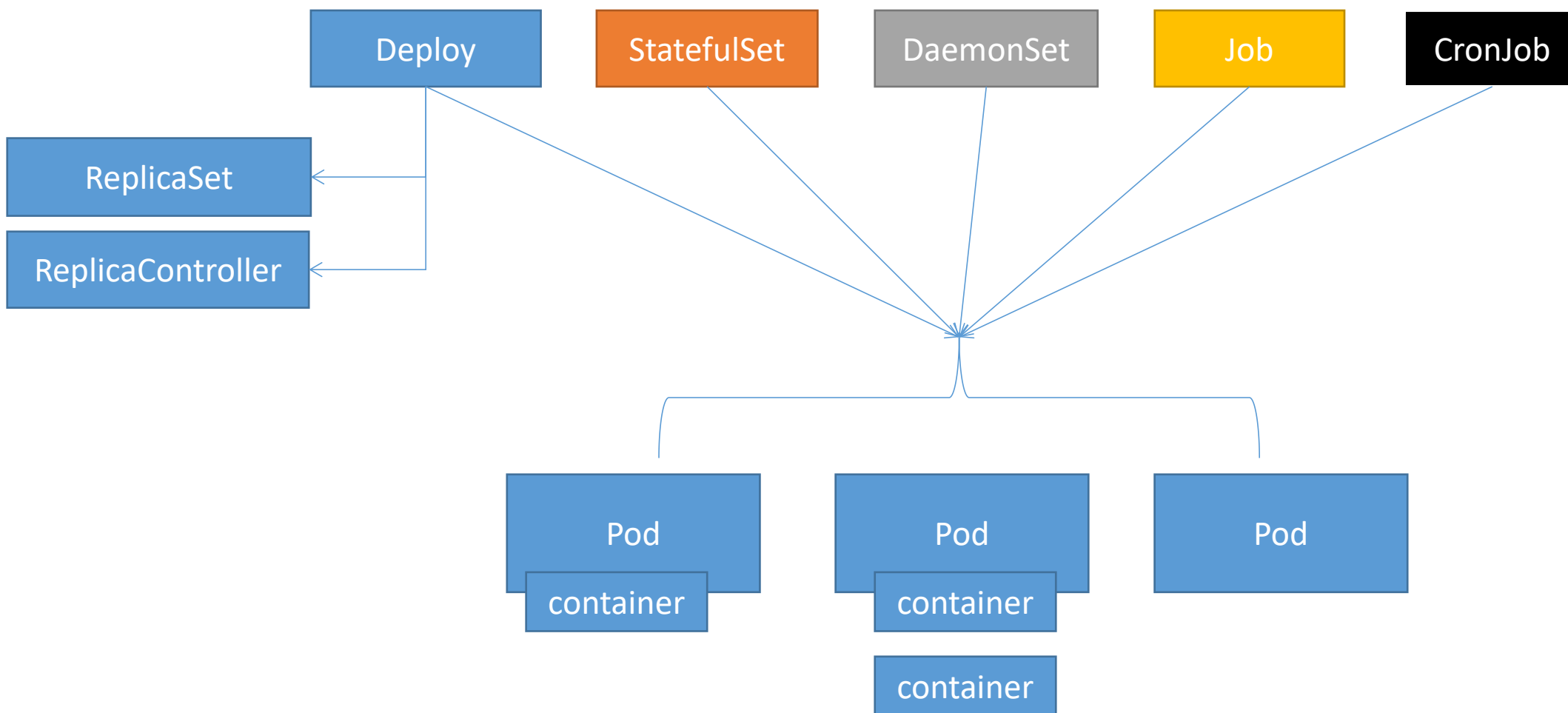
Pod



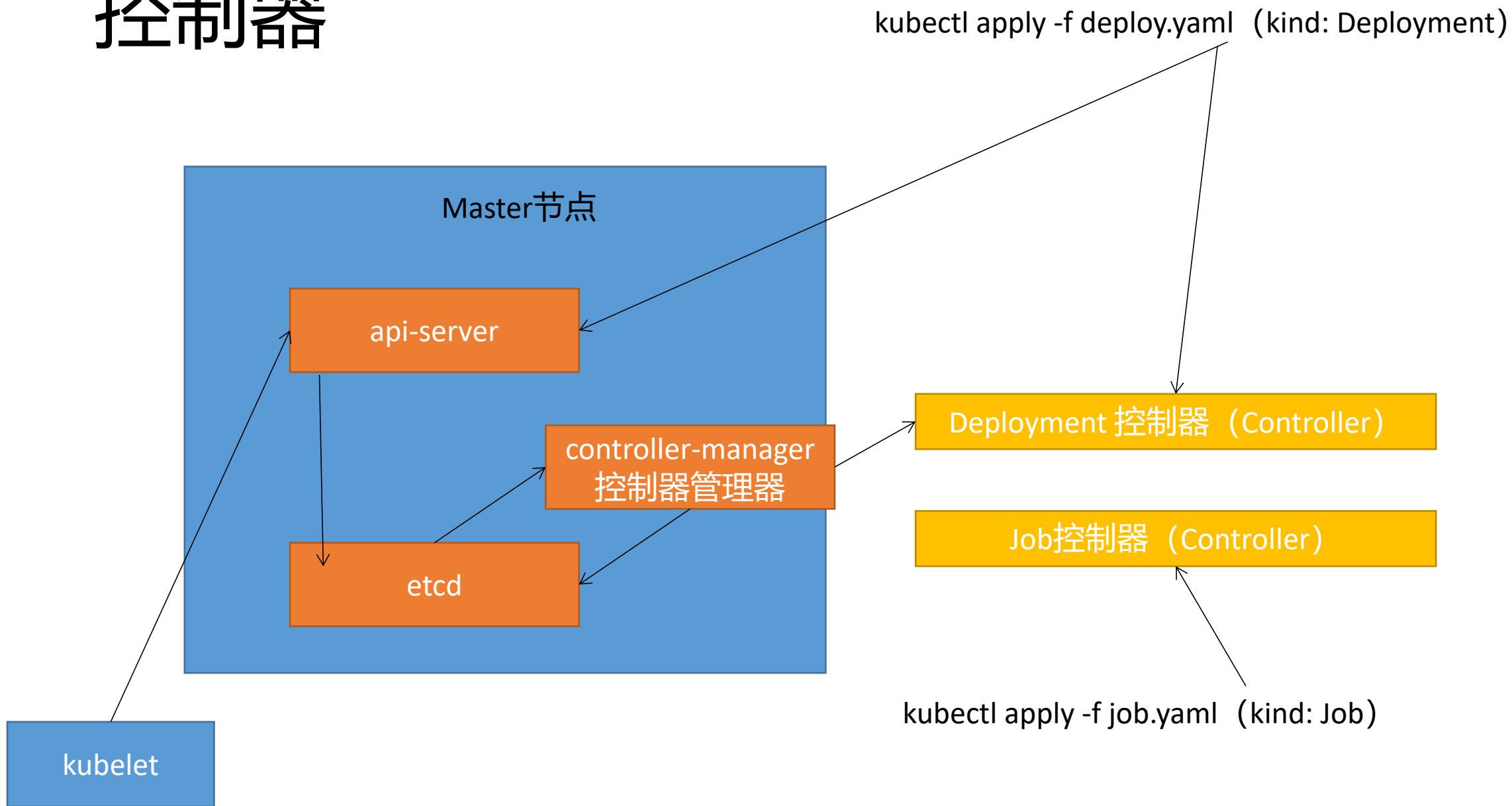
Pod



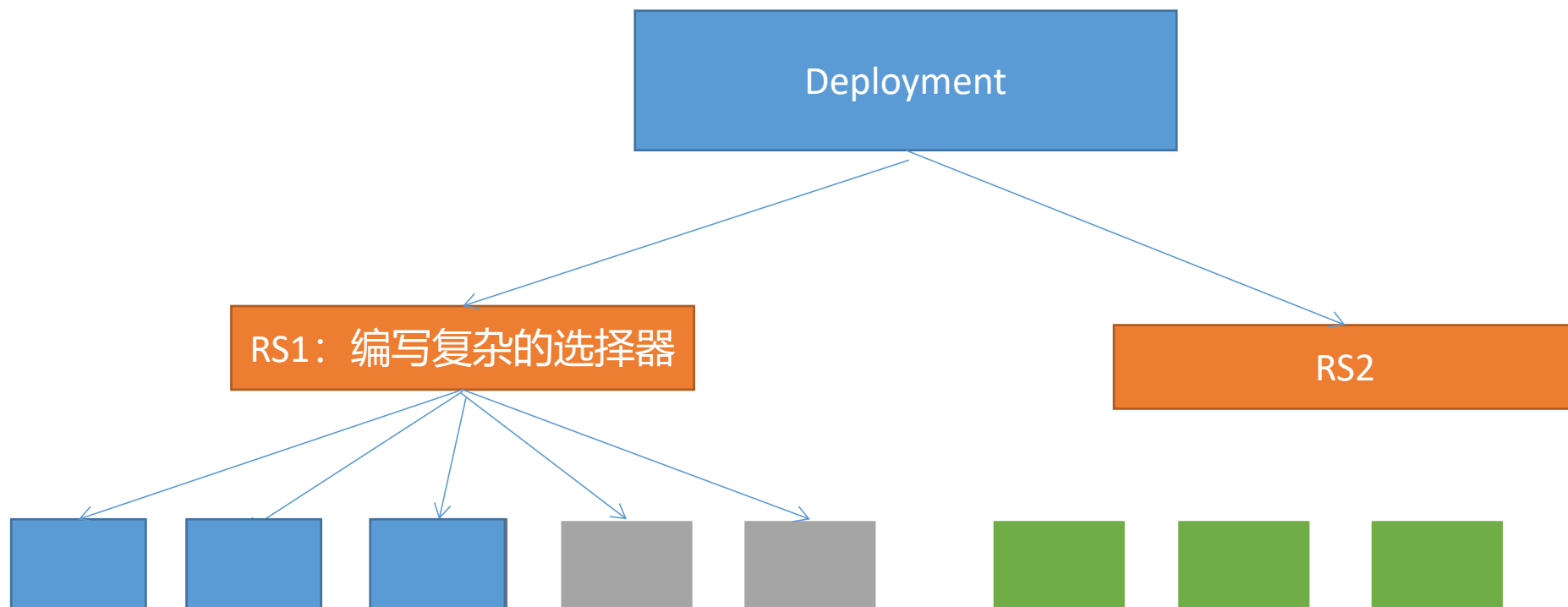
工作负载



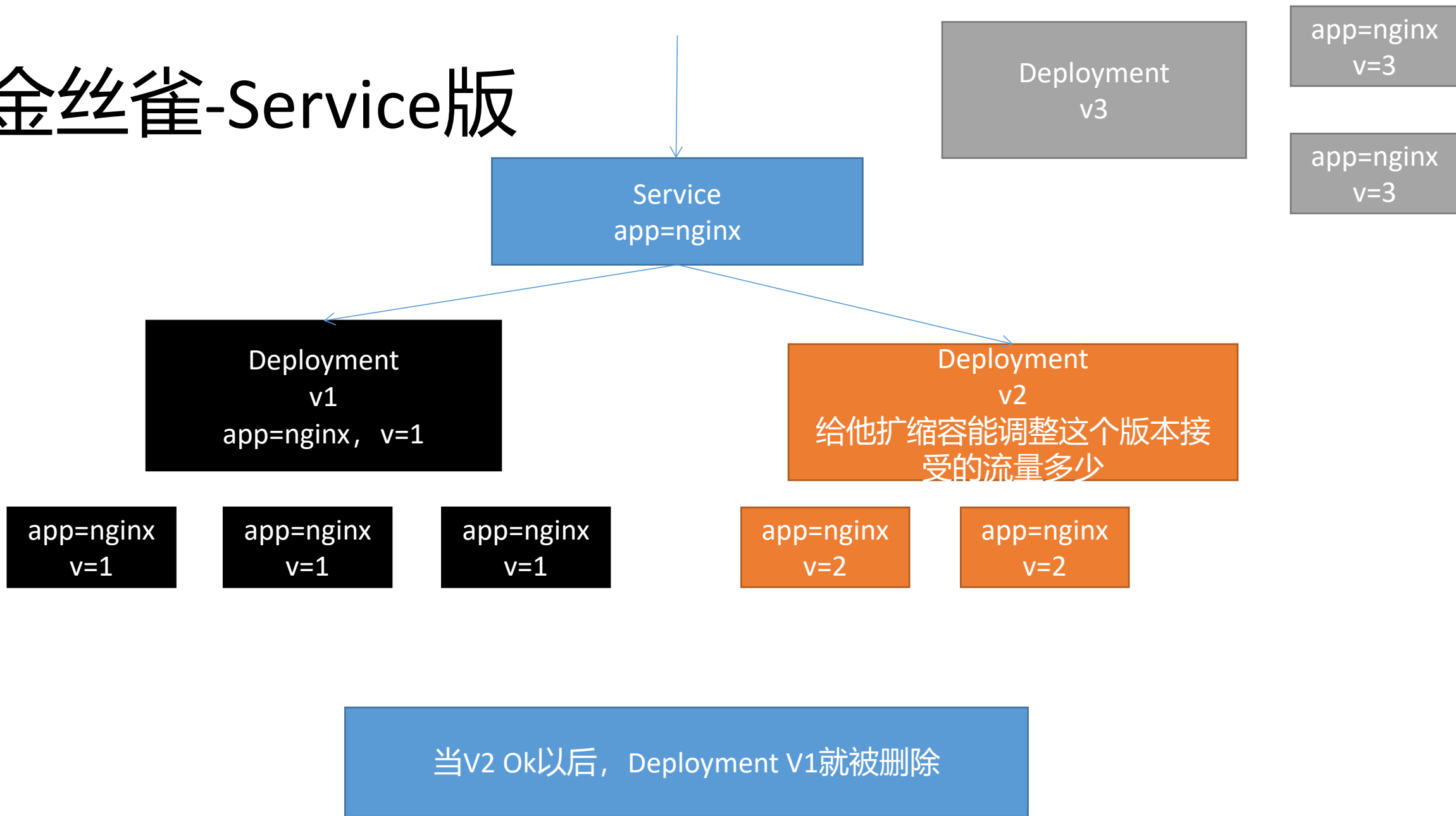
控制器



滚动更新



金丝雀-Service版

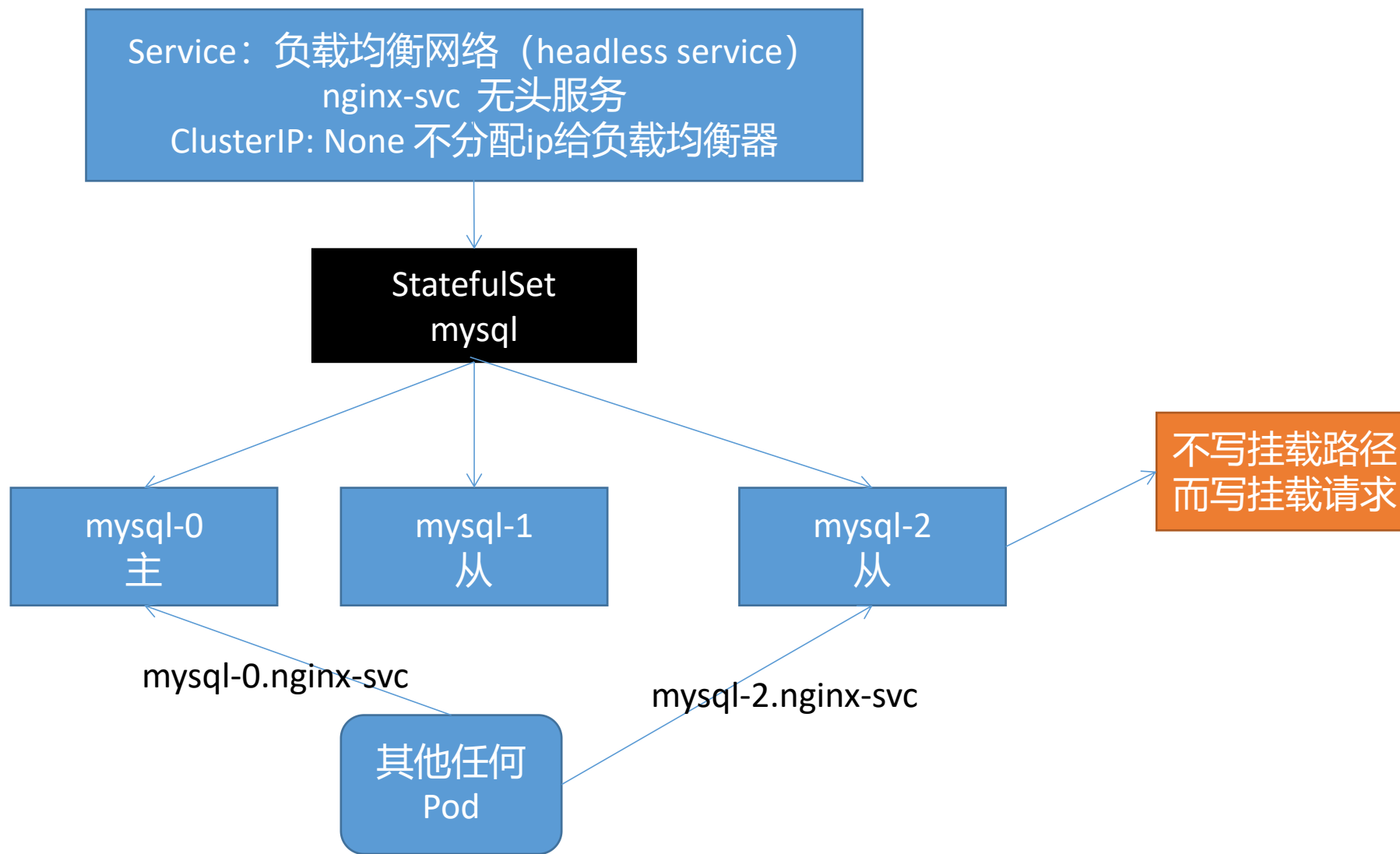


StatefulSet

全地址

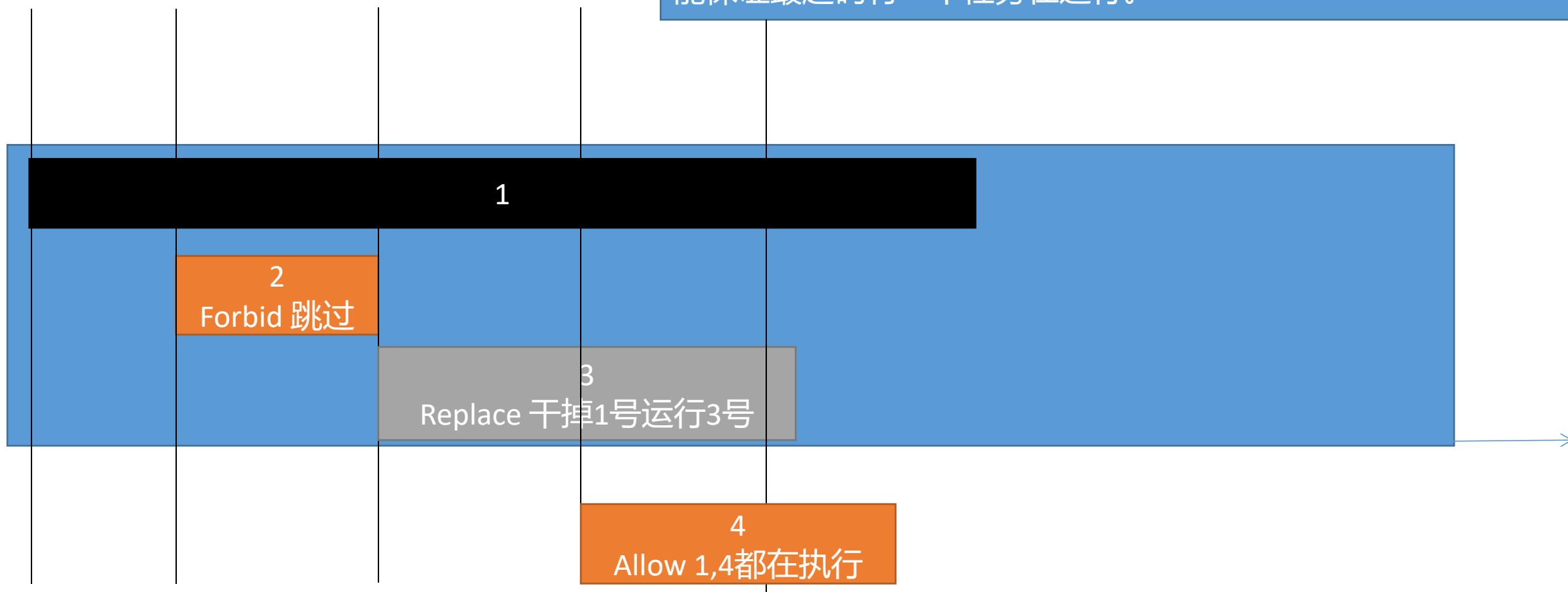
pod-specific-string.serviceName.default.svc.cluster.local

pod名.service名.namespace名.后面一串默认的



CronJob

startingDeadlineSeconds: 启动的超时时间 600s。设置超大
concurrencyPolicy: 并发策略。设置为Allow "Allow" (允许, default)
"Forbid"(禁止): forbids; 前个任务没执行完, 要并发下一个的话,
个会被跳过
"Replace"(替换): 新任务, 替换当前运行的任务
能保证最起码有一个任务在运行。



Service整个端口问题

curl 10.170.11.11:6379 访问不到

```
port: 80
targetPort: 8080
```

Service: 10.170.11.88
cluster-service-02

port: 80

port: 99

Service: 10.170.11.11
cluster-service-test

port: 80

port: 99

```
- name: abc
  port: 80
  targetPort: 8080
- name: redis
  port: 99
  targetPort: 6379
```

targetPort: 8080

container-03 不能占用8080

Pod: 也有ip。只要有ip就认为是一个新主机

app: canary-tomcat

tomcat-container

containerPort: 8080

redis-container

containerPort: 6379

targetPort: 8080

Pod

app: canary-tomcat

targetPort: 8080

Pod

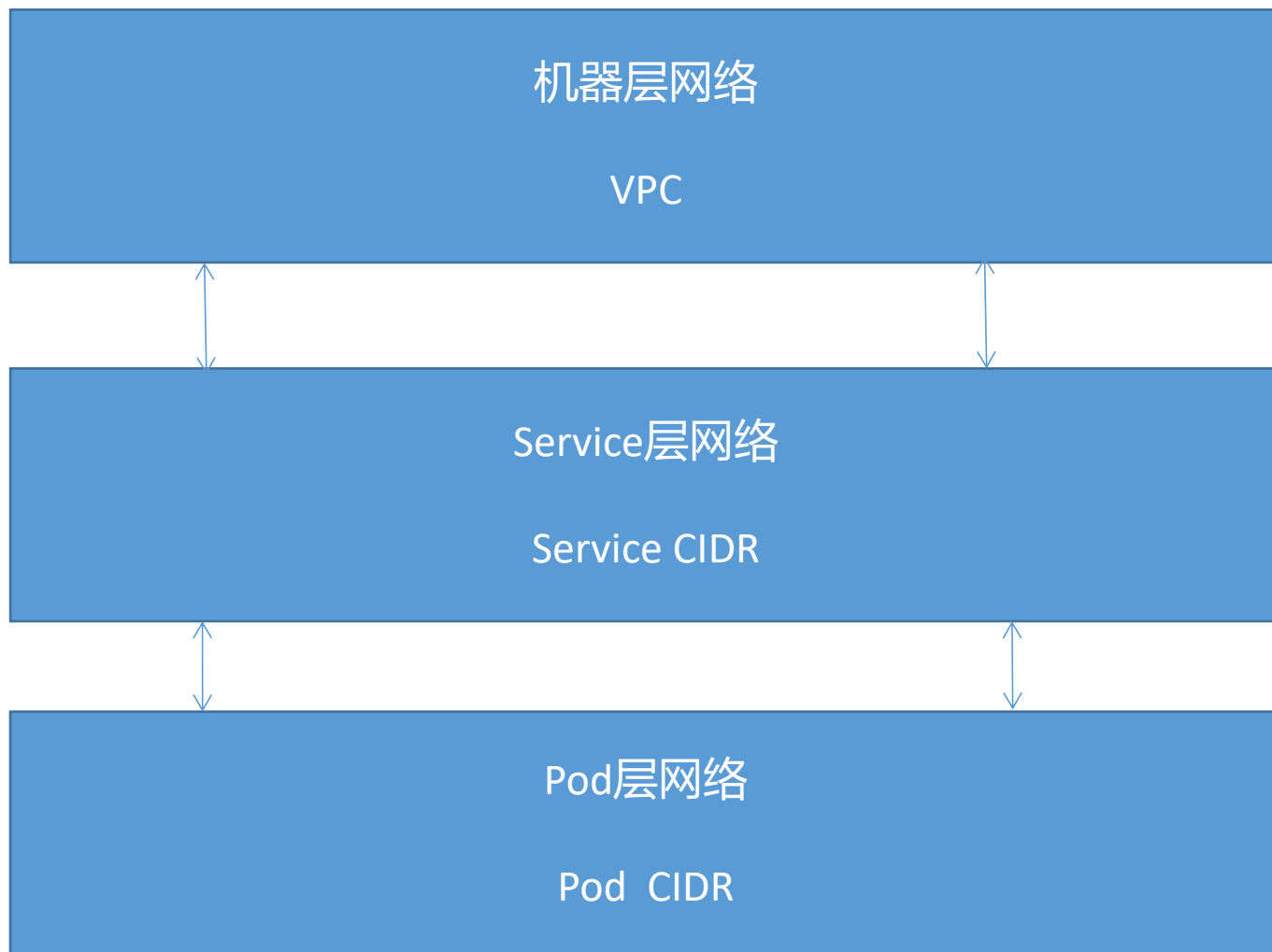
app: canary-tomcat

curl 10.170.11.11:80

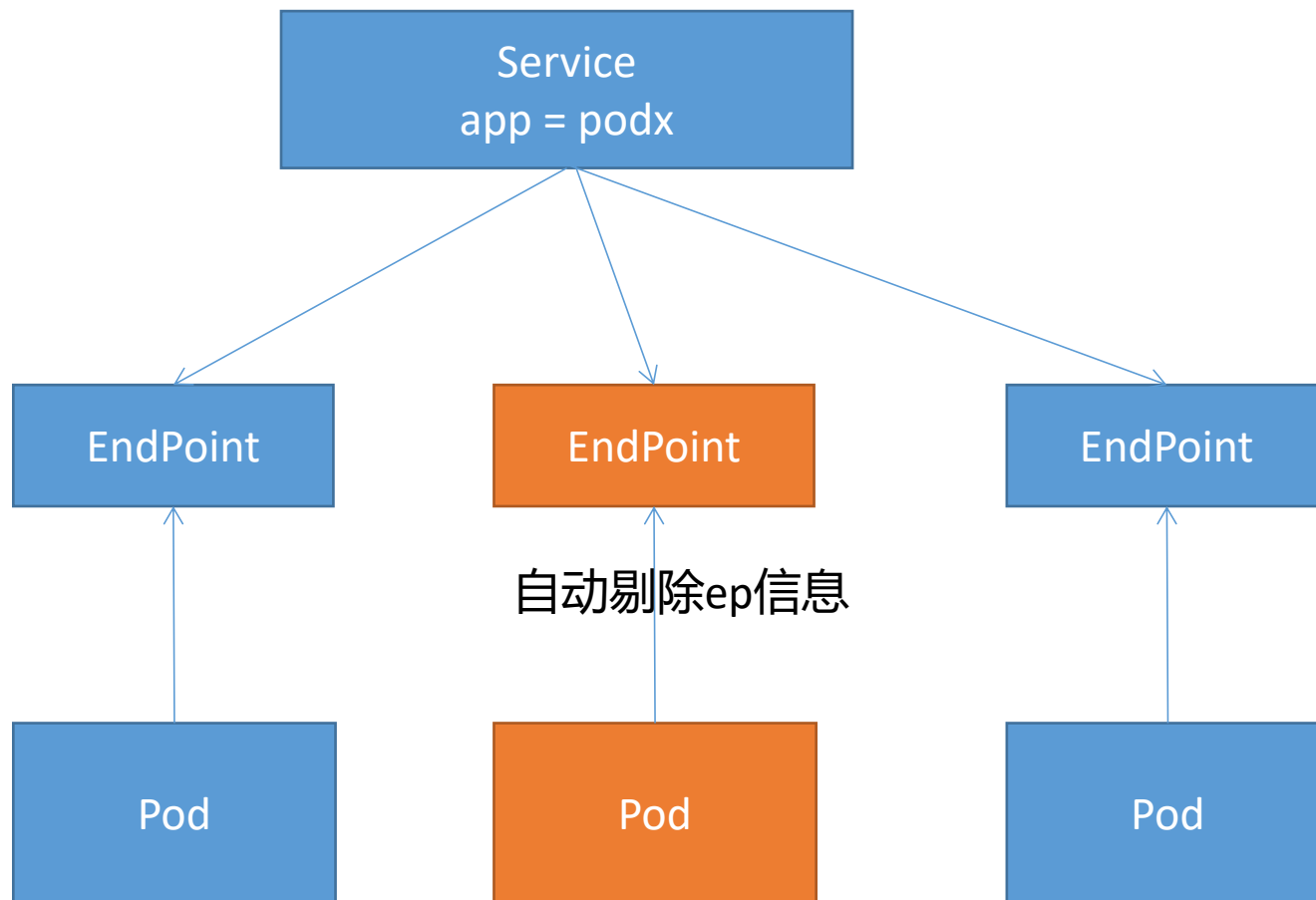
```
port: 80
targetPort: 8080
```

以上所有端口和Node的端口没有任何冲突

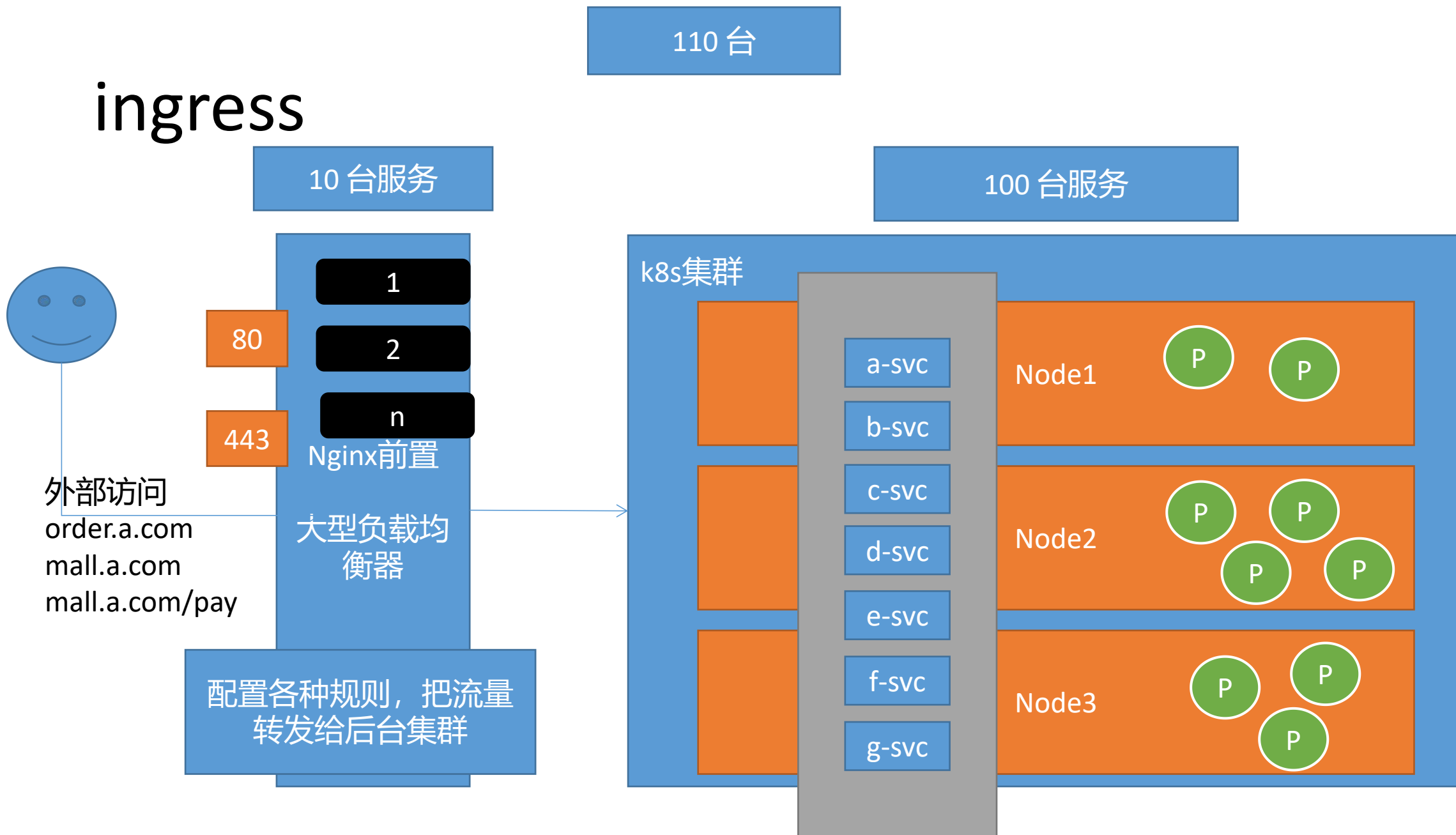
网络层次 -- 默认全是通的

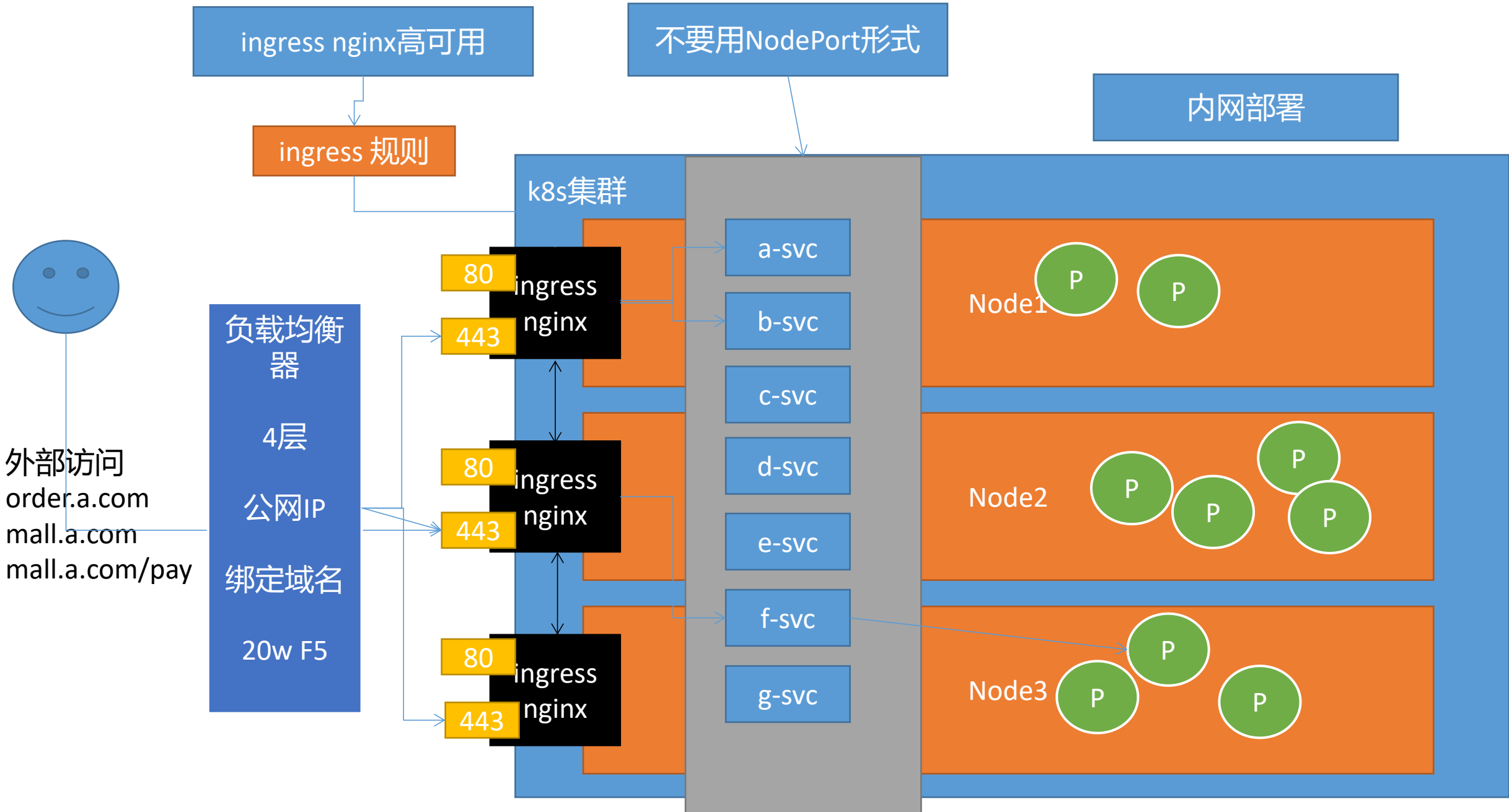


Service原理

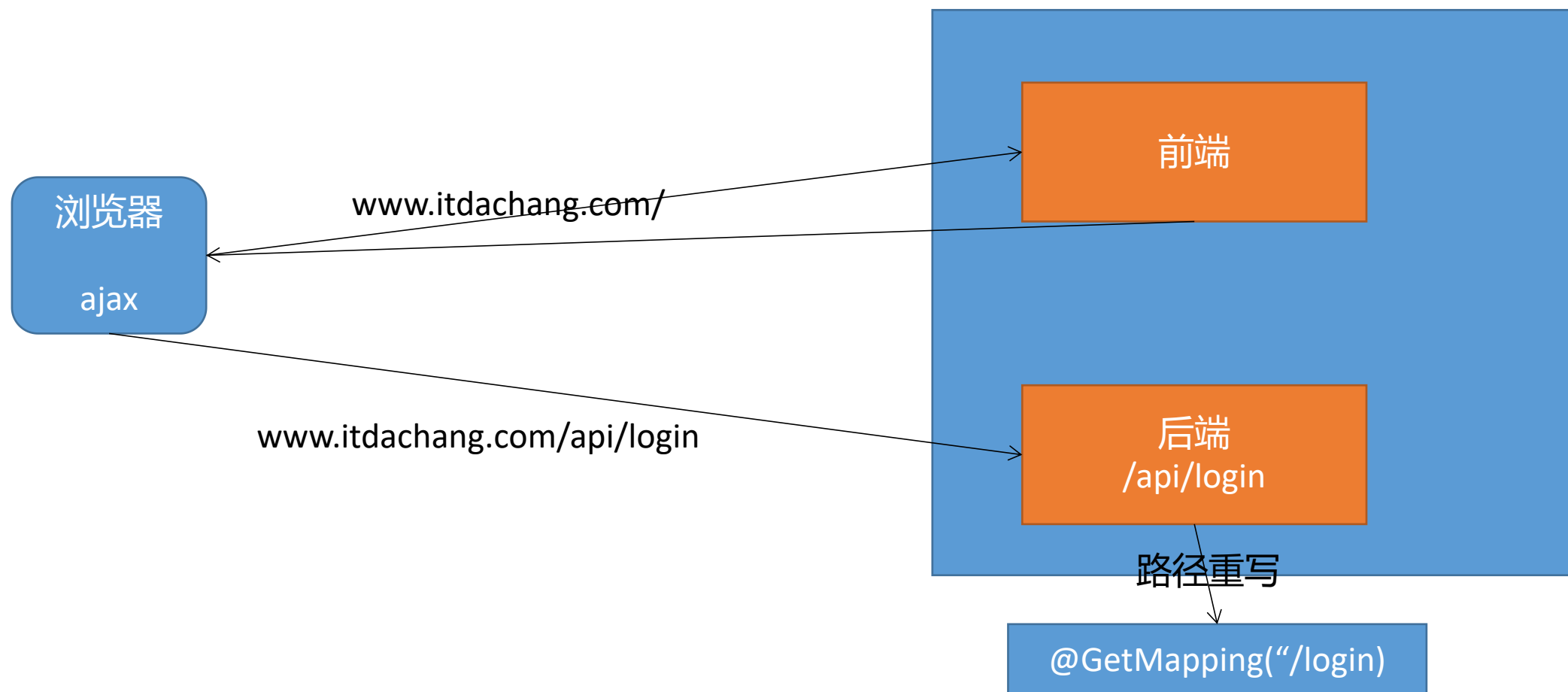


ingress

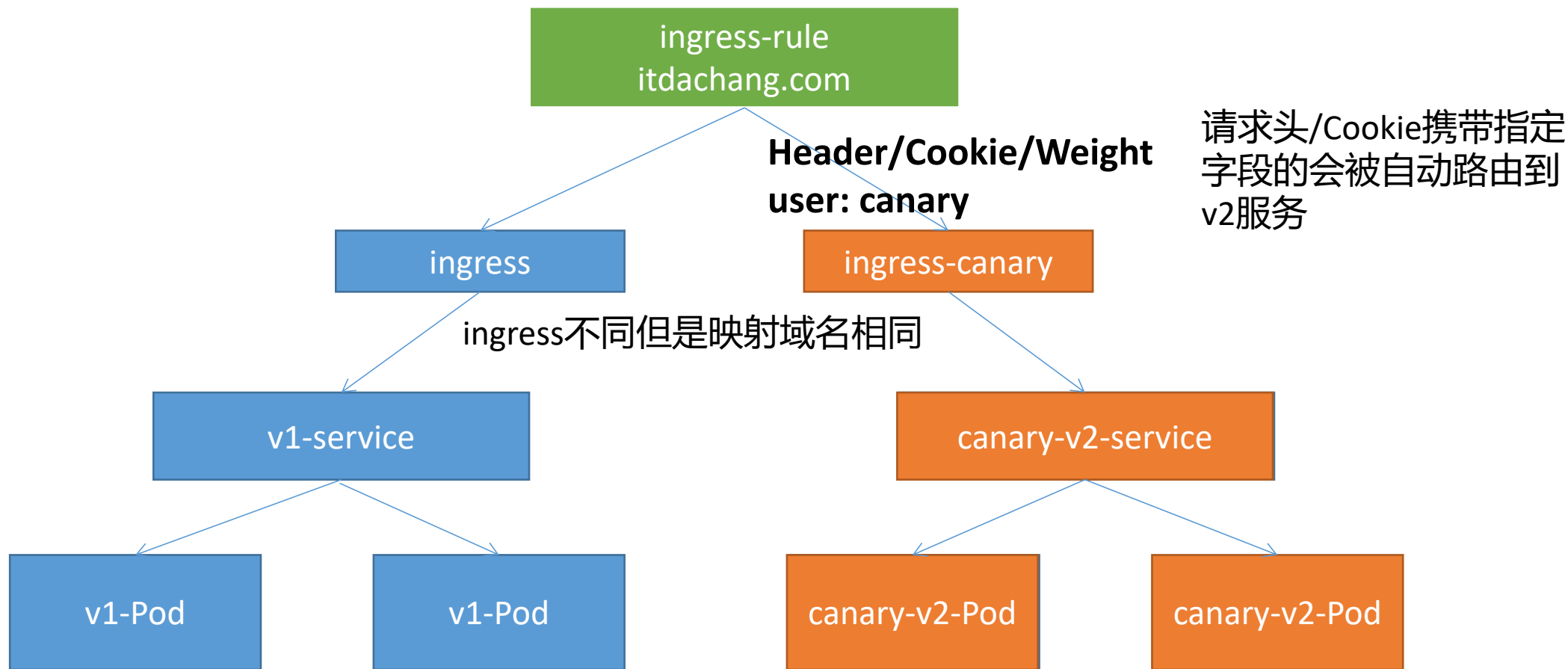




路径重写

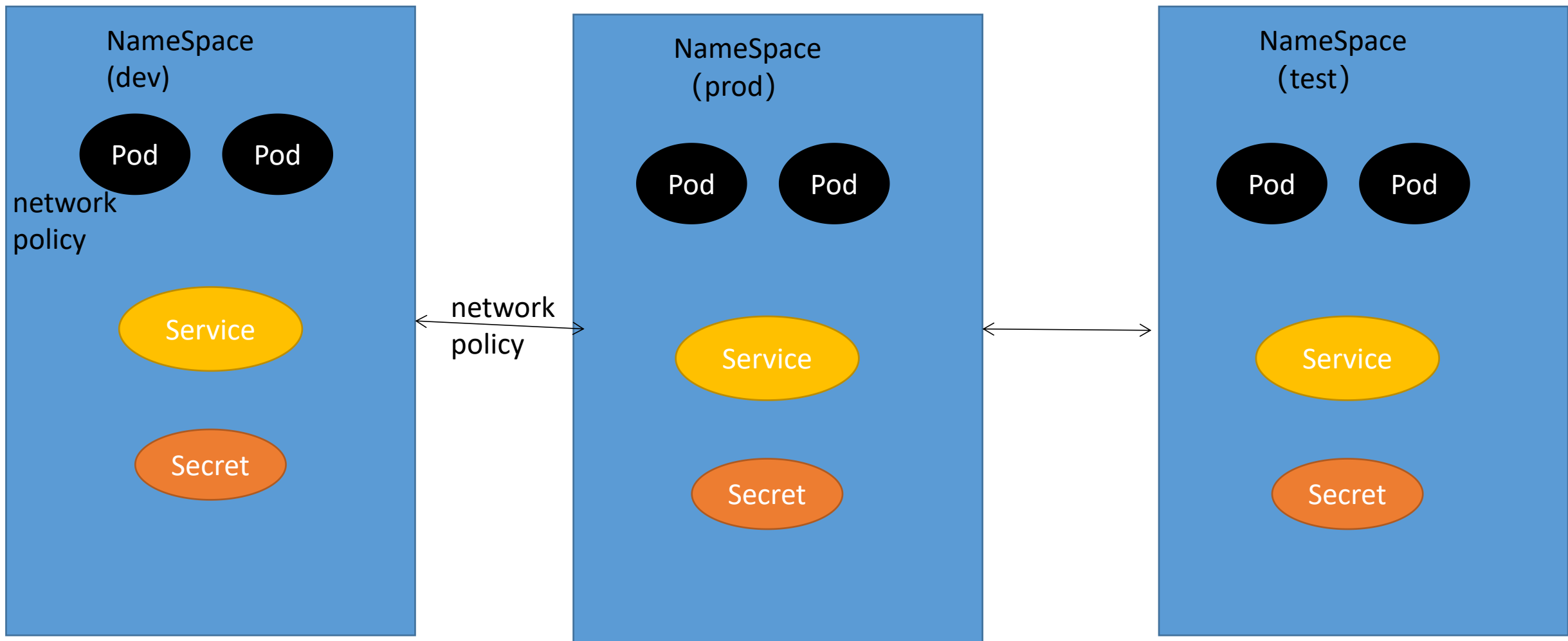


金丝雀-Ingress版



以后新版本上线，配置新的ingress-canary规则即可。
canary验证通过以后，移除旧的ingress和service。
取消当前ingress-canary的annotation，变为普通的ingress

NetworkPolicy-网络互通性



CM与SpringBoot

做到开发人员无感知生产环境的核心配置

SpringBoot
开发环境

application.yaml

application-dev.yaml

application-prod.yaml
数据库的账密信息

SpringBoot

SpringBoot (jar包也会放在容器的/app下)
上云
deploy.yaml
volumeMounts:
 name: prod-conf
 mountPath: /app
volumes:
 name: prod-conf
 configMap:
 name: mall-conf

cm: mall-conf
data:
 application.yaml: |
 生产环境的所有配置

Pod
/app application.yaml
/app xxx.jar
java -jar xxx.jar
SpringBoot启动默认行为让外部的yaml优先

Nginx-可以使用子路径的方式

```
conf
  nginx.conf
  conf.d
  xxx
  xxx
```

```
nginx.conf
mount:
  path: /etc/nginx
  subPath: nginx.conf
```

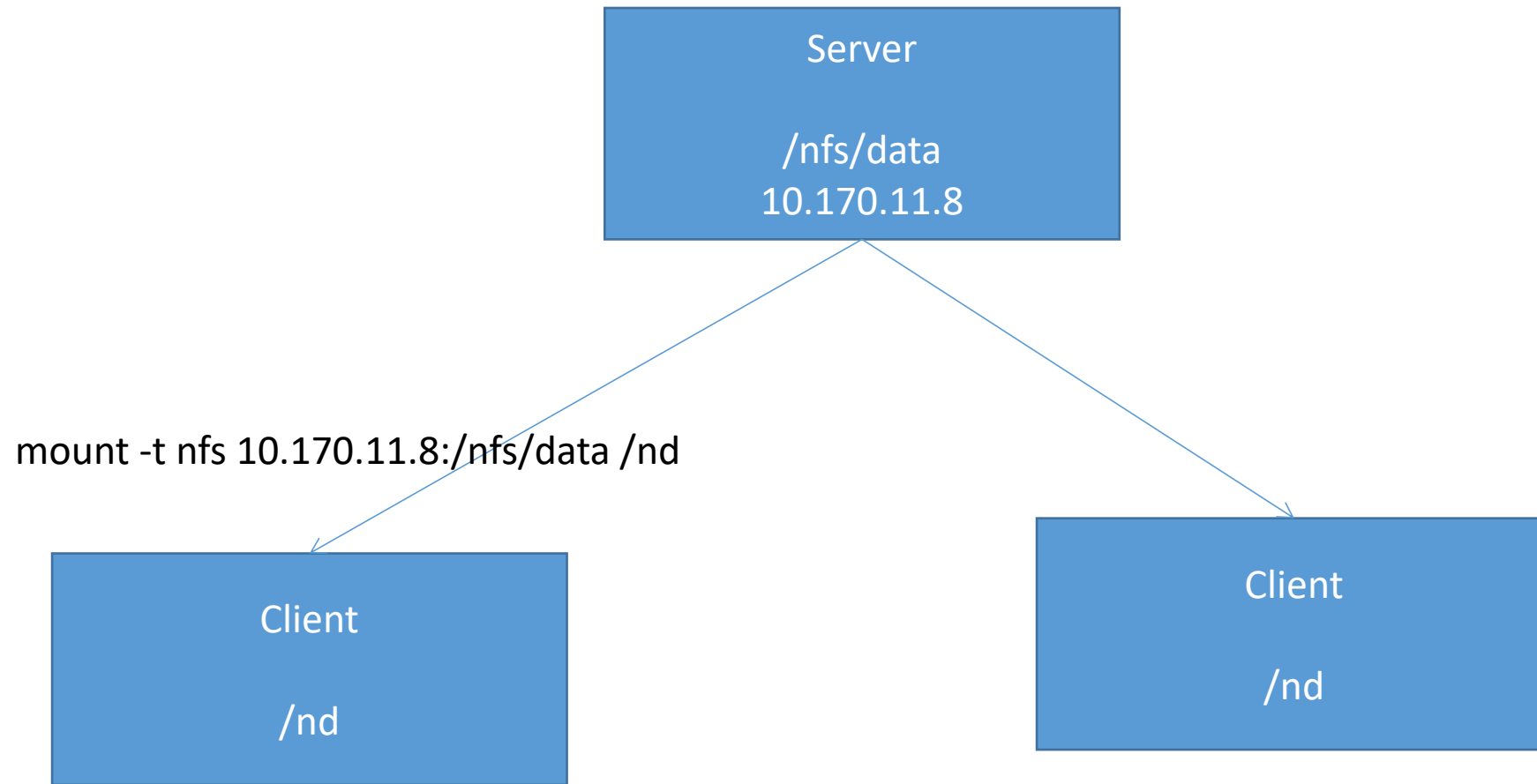
```
nginx-dir
mount:
  path: /etc/nginx
  subPath: conf.d
```

conf.d

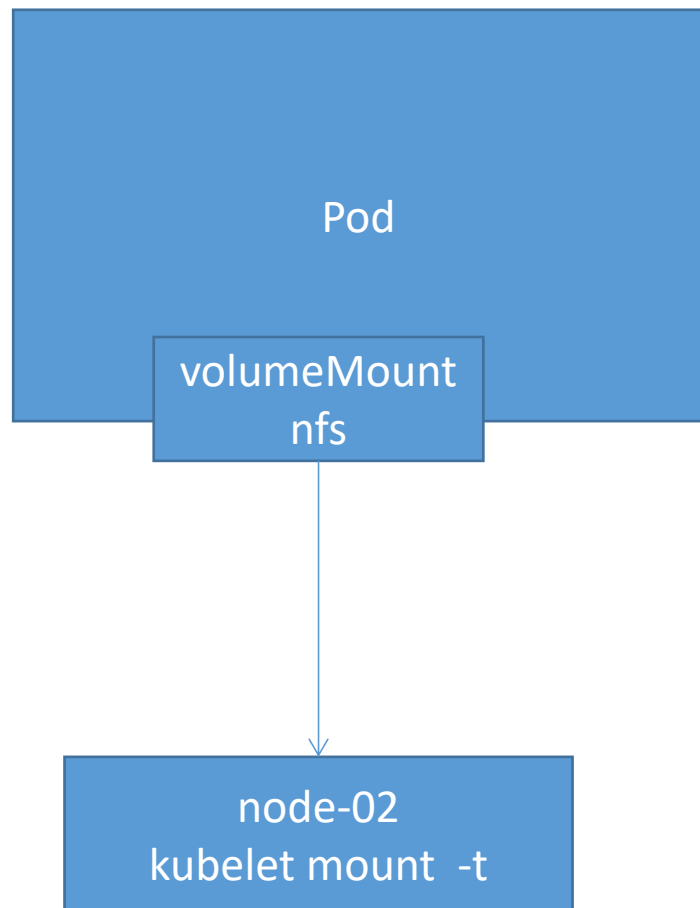
volumes:

- name: nginx-conf
hostPath:
path: /app/nginx/nginx.conf
type: File
- name: nginx-dir
hostPath: ## 主机的这个文件
path: /app/nginx/conf.d
type: Directory

NFS



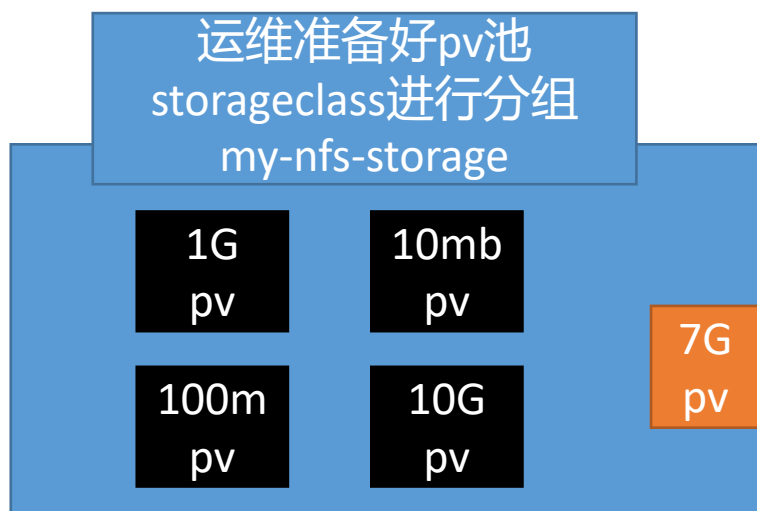
直接挂载



痛点

- Pod的开发人员很清楚容器的哪些位置适合挂载什么
- 开发人员并不清楚存储技术。存储还要编写详细文档
- 要求：Pod文件必须描述每个挂载该怎么挂

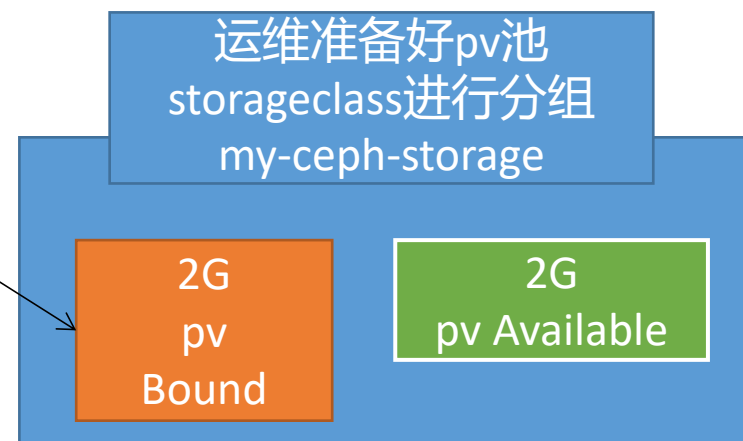
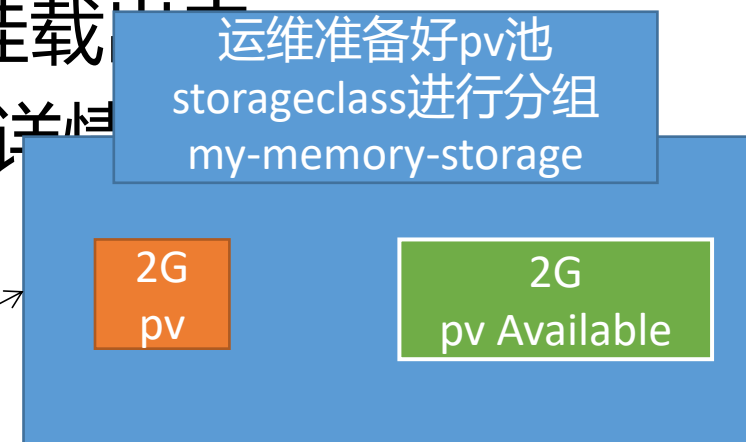
缺点：资源浪费



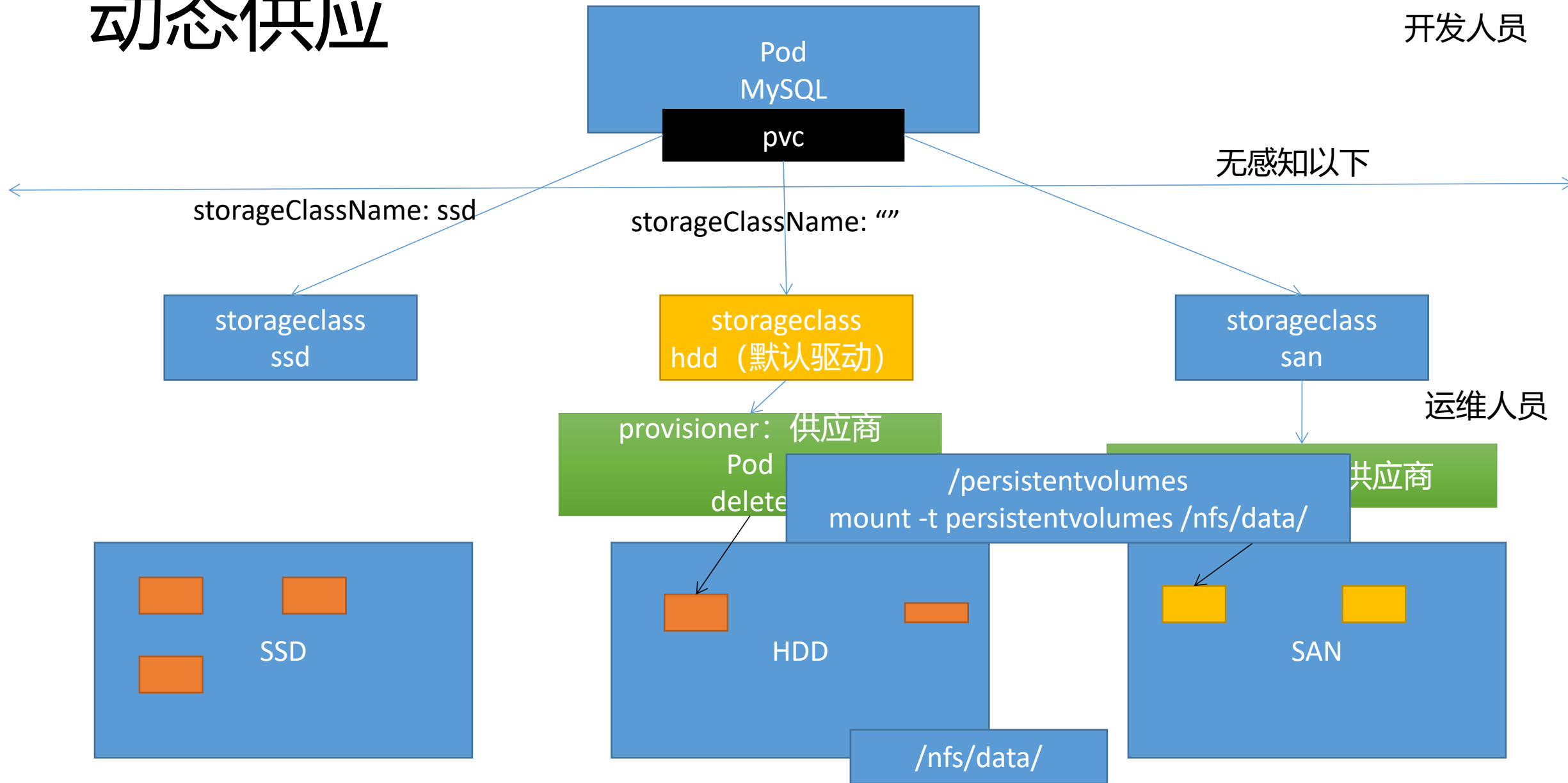
Pod
自己绑定的申请书自己用，
空间自己用。默认别人都
不能挂，即使pvc删除

pvc 2g

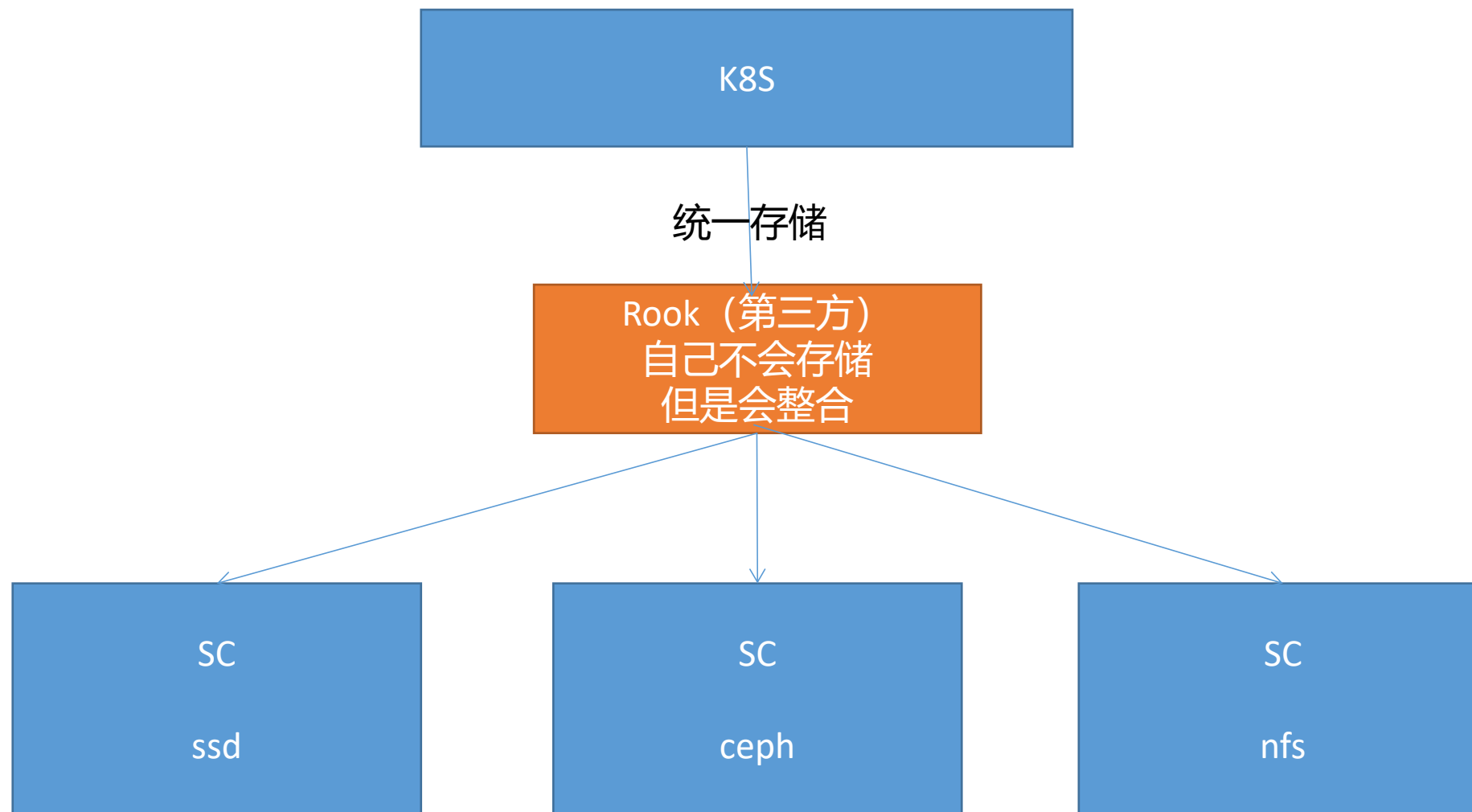
静态供应



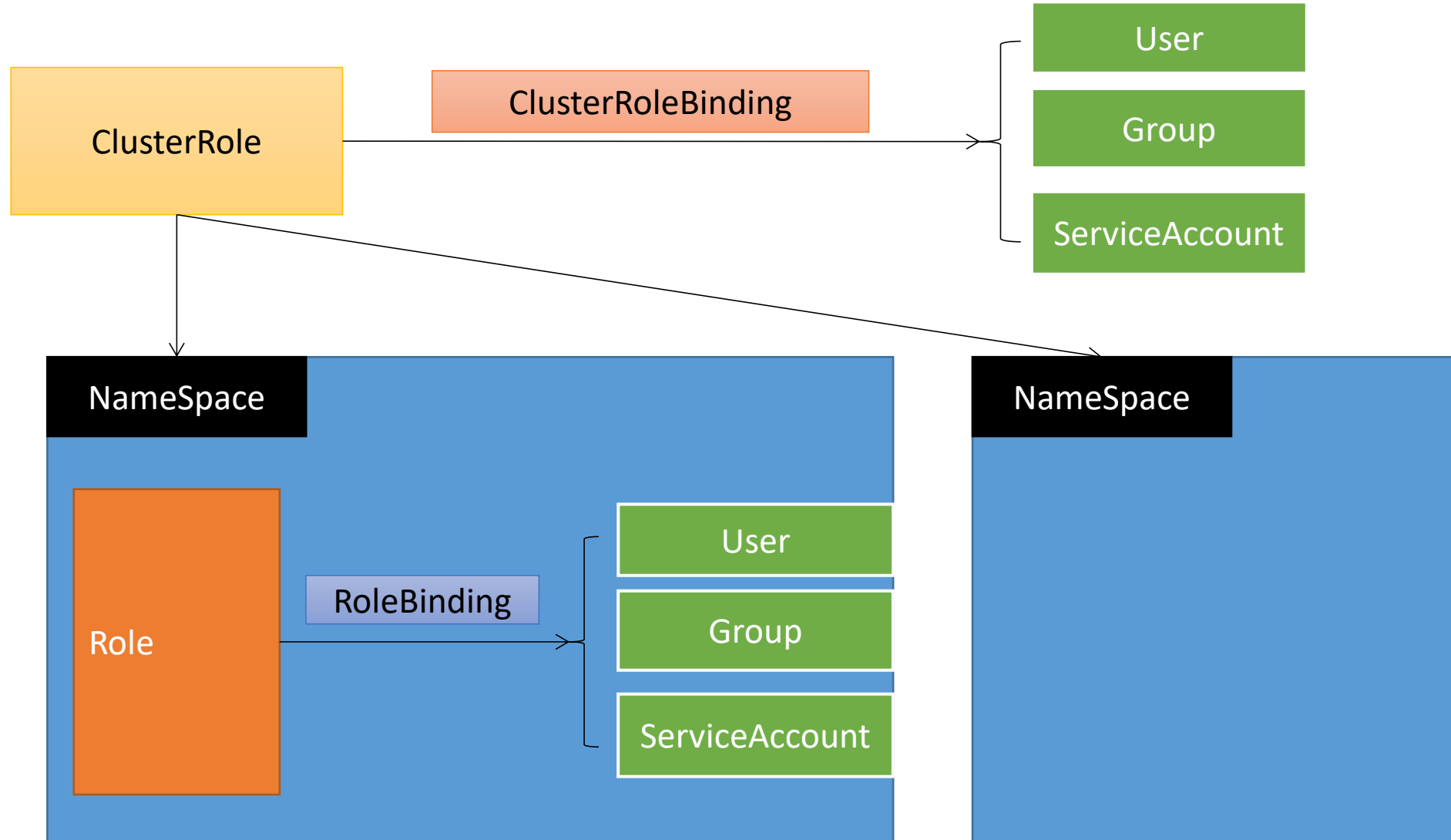
动态供应



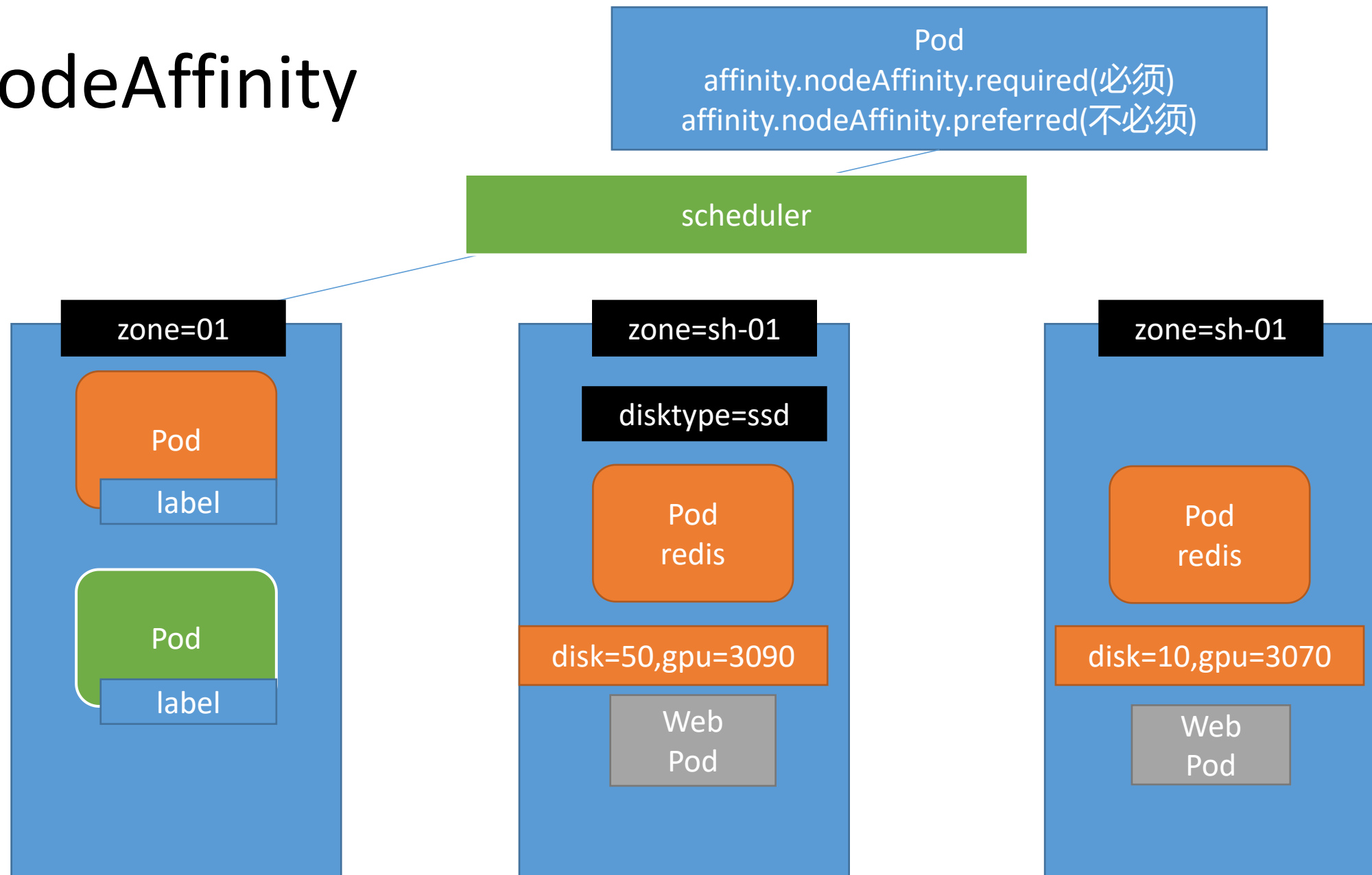
动态供应



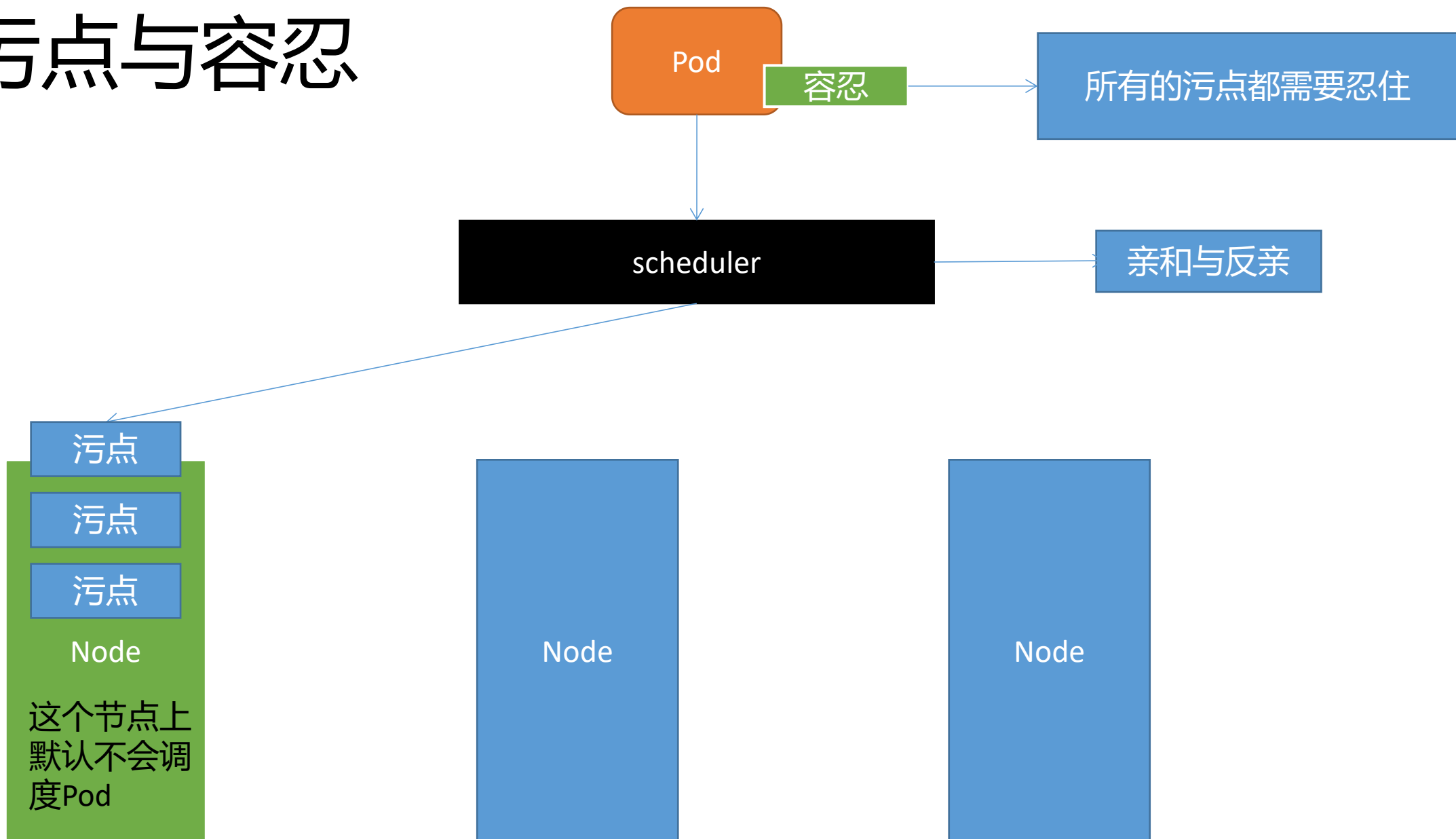
RBAC



nodeAffinity

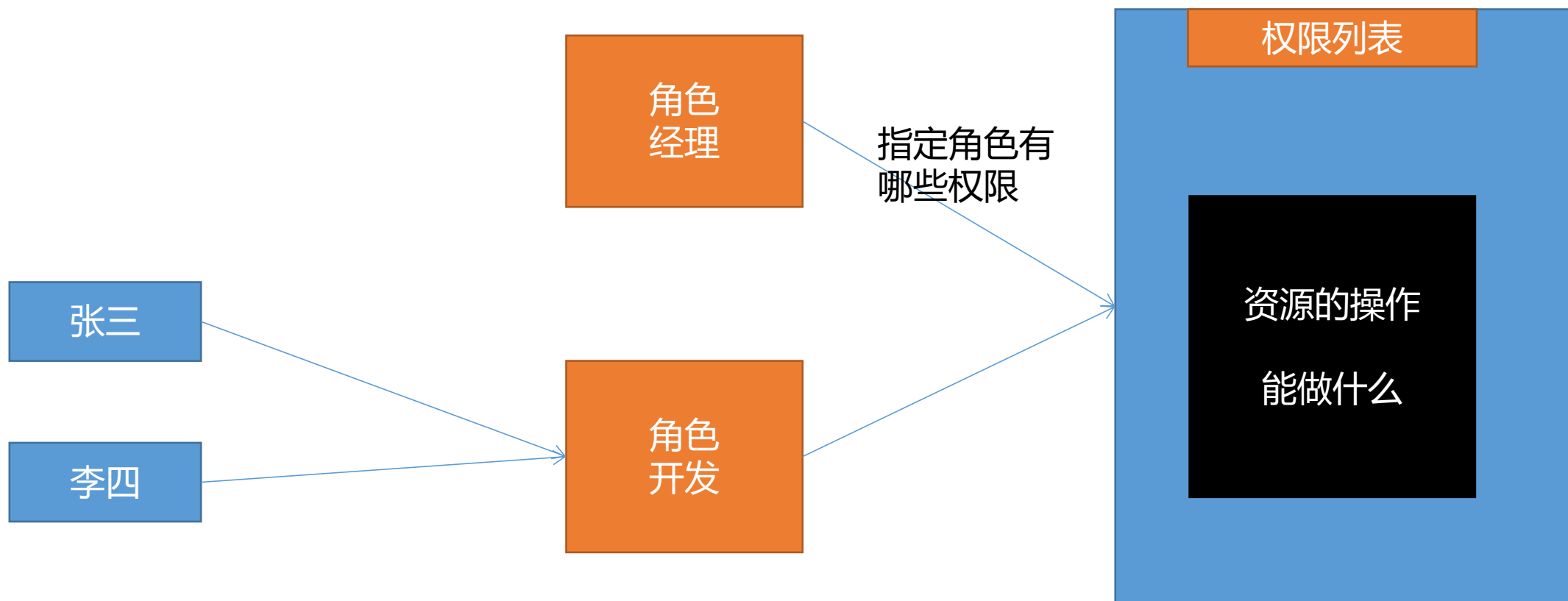


污点与容忍

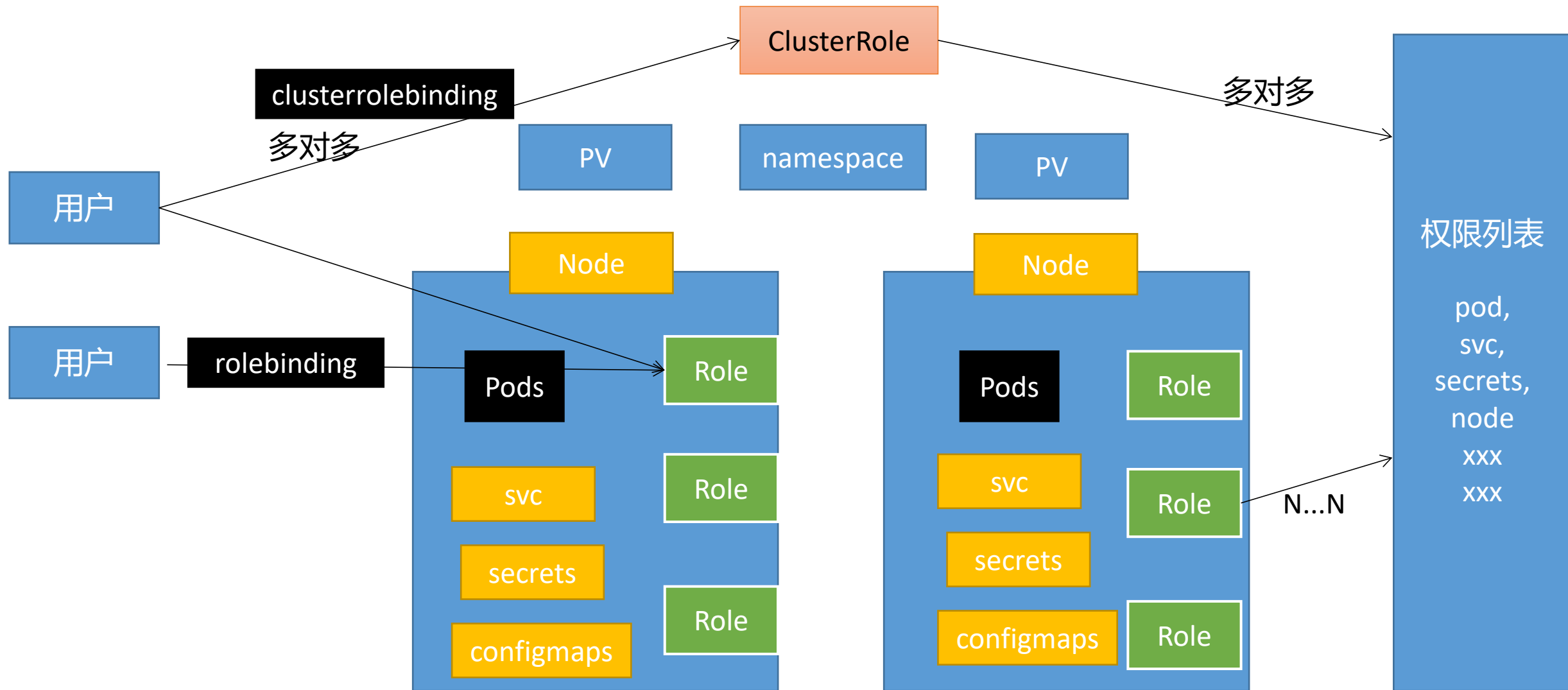


RBAC (Role-Based-Access-Controller)

- 基于角色的访问控制

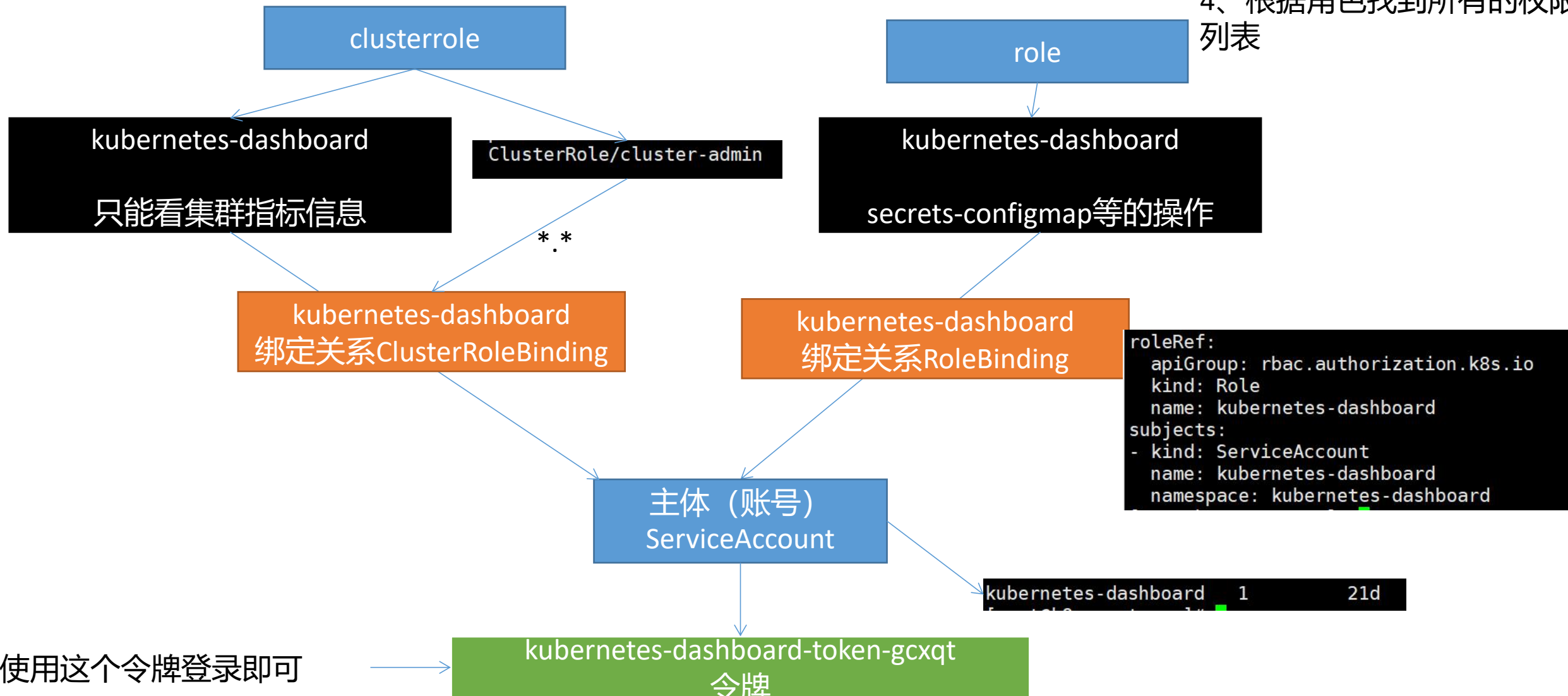


k8s-Role



Dashboard为什么能操作整个集群

- 1、每个账号一个令牌
- 2、带着令牌可以找到账号
- 3、根据账号找到所有绑定的角色
- 4、根据角色找到所有的权限列表



RBAC是这样

- 1、自己创建一个账号 ServiceAccount
 - 创建的账号，默认会关联一个secret（秘钥信息）令牌
- 2、ServiceAccount来绑定一些角色
 - 一个sa是一个账号
 - Pod会自动挂载名称空间下默认的账号
- 3、使用账号的令牌登录，就能知道这个账号的权限

Helm

Charts

本地Helm
添加Repo

仓库位置1

仓库位置2

```
helm repo add bitnami https://charts.bitnami.com/bitnami
helm search repo bitnami mysql
helm search hub mysql
```

```
helm repo update #更新仓库索引
```

Repository
<https://artifacthub.io/>

bytedance repo

atguigu repo

alibaba repo