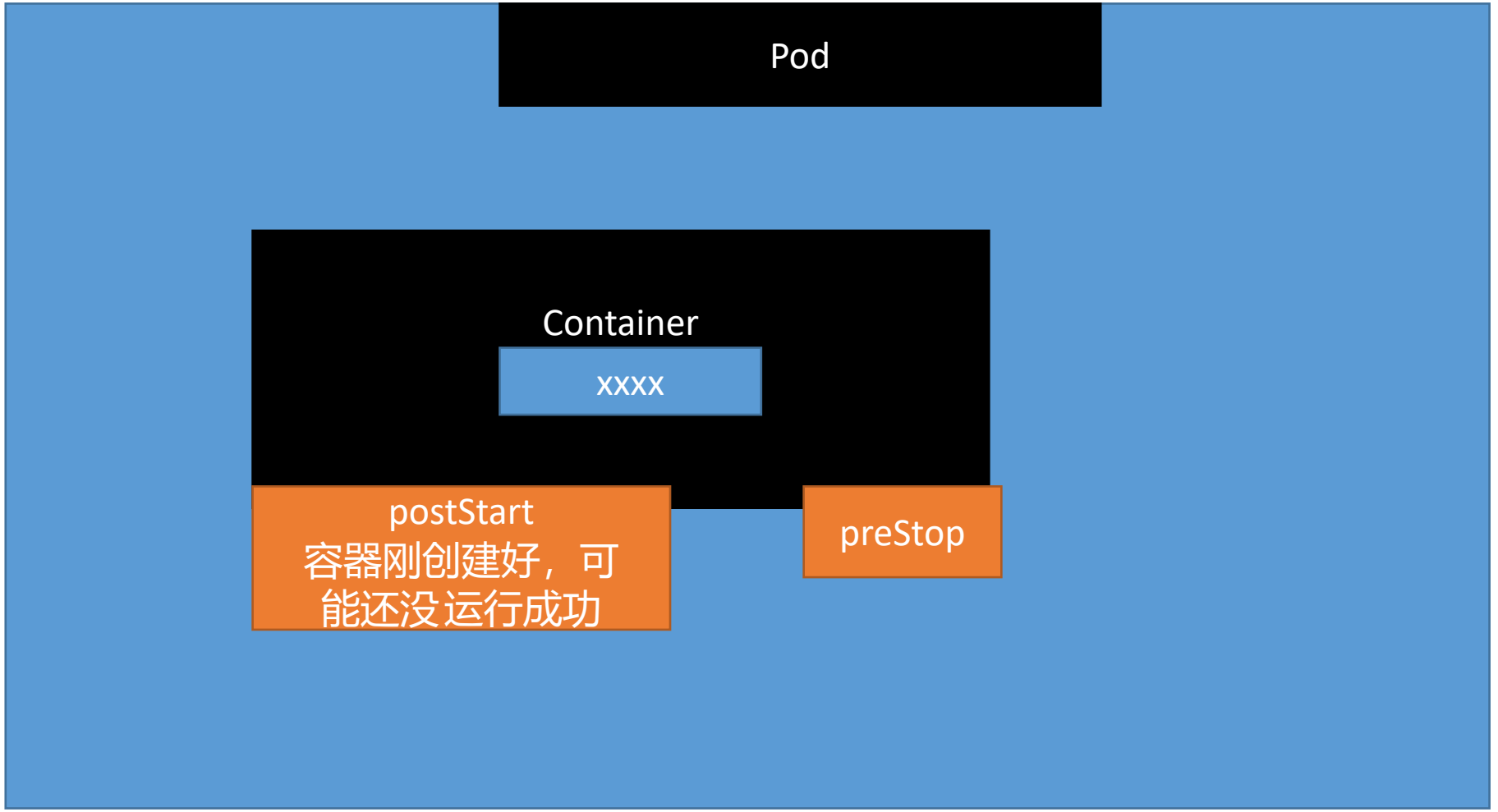
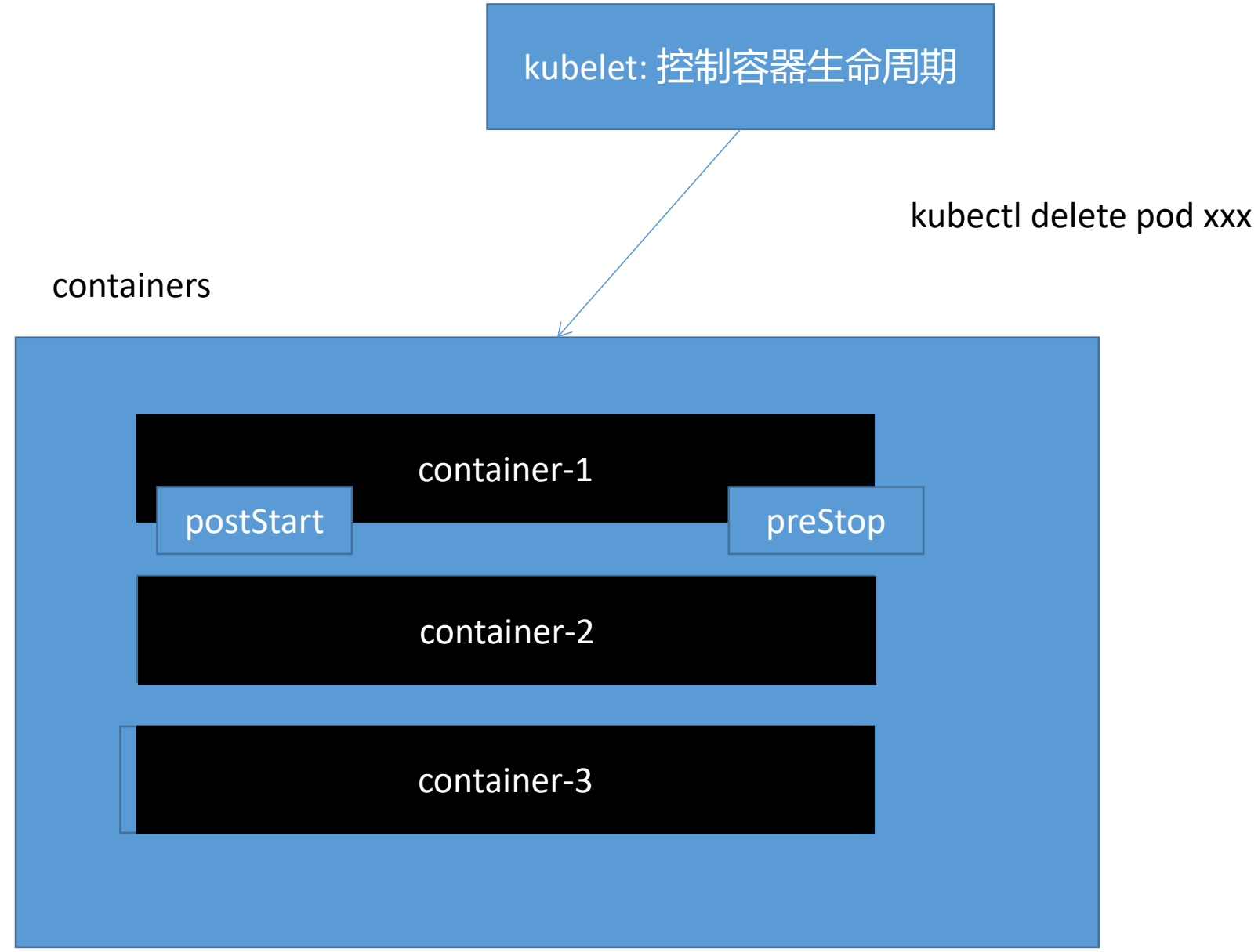
K8S图例

容器



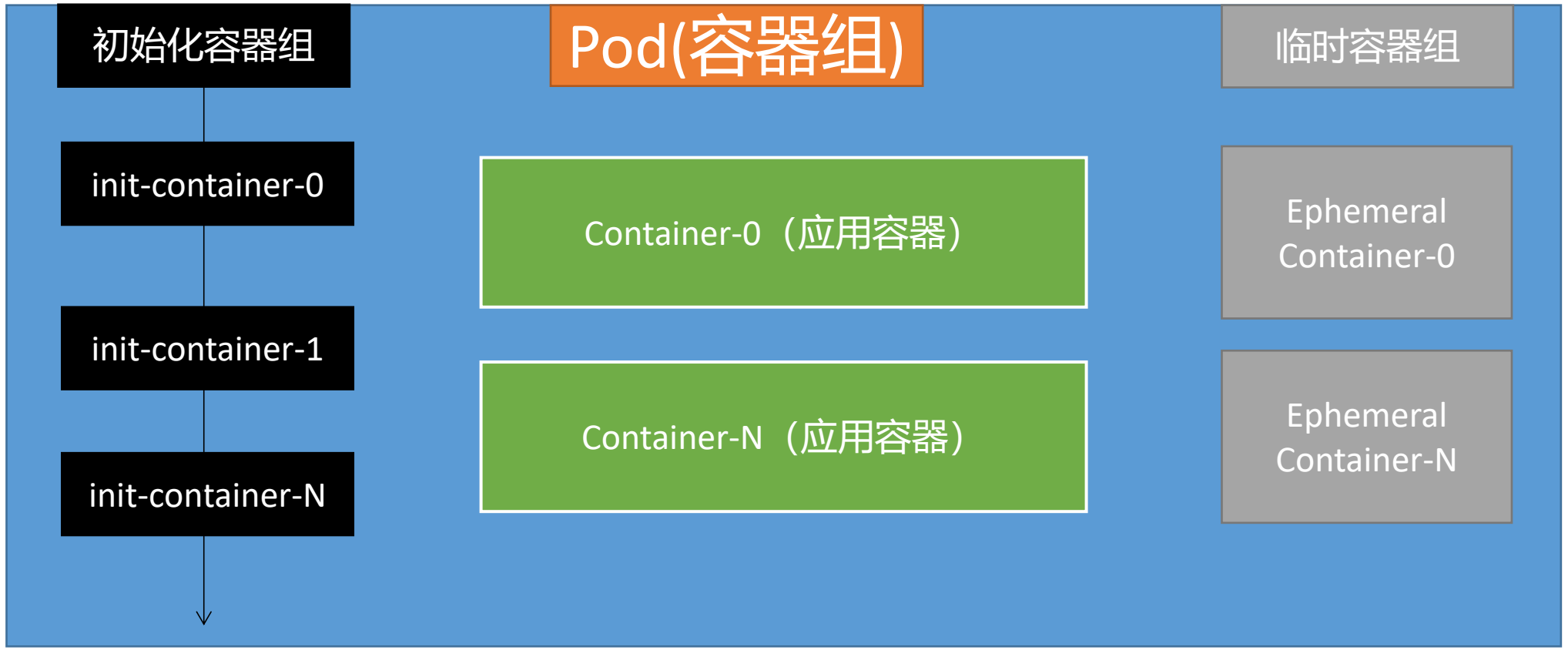
Pod



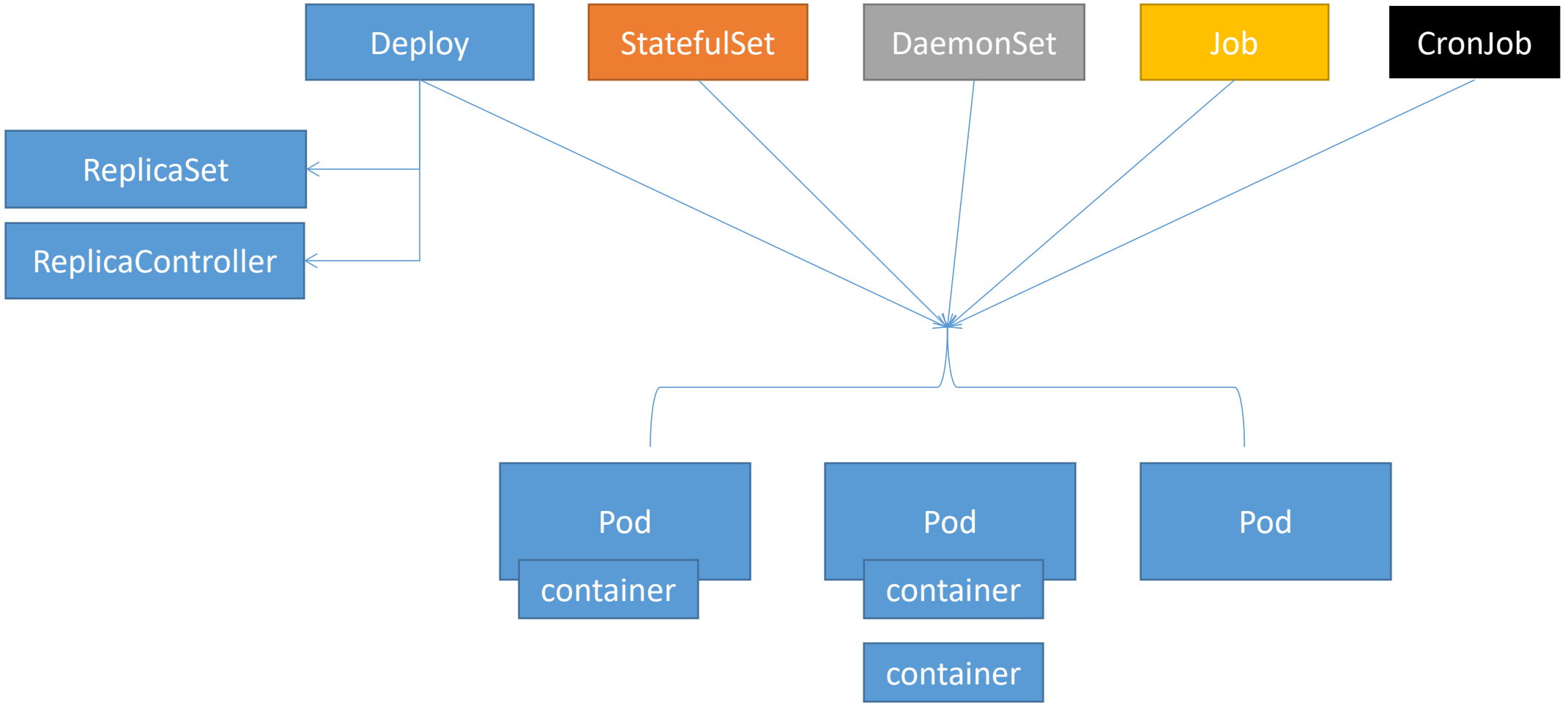
postStart:创建之后

preStop:停止之前

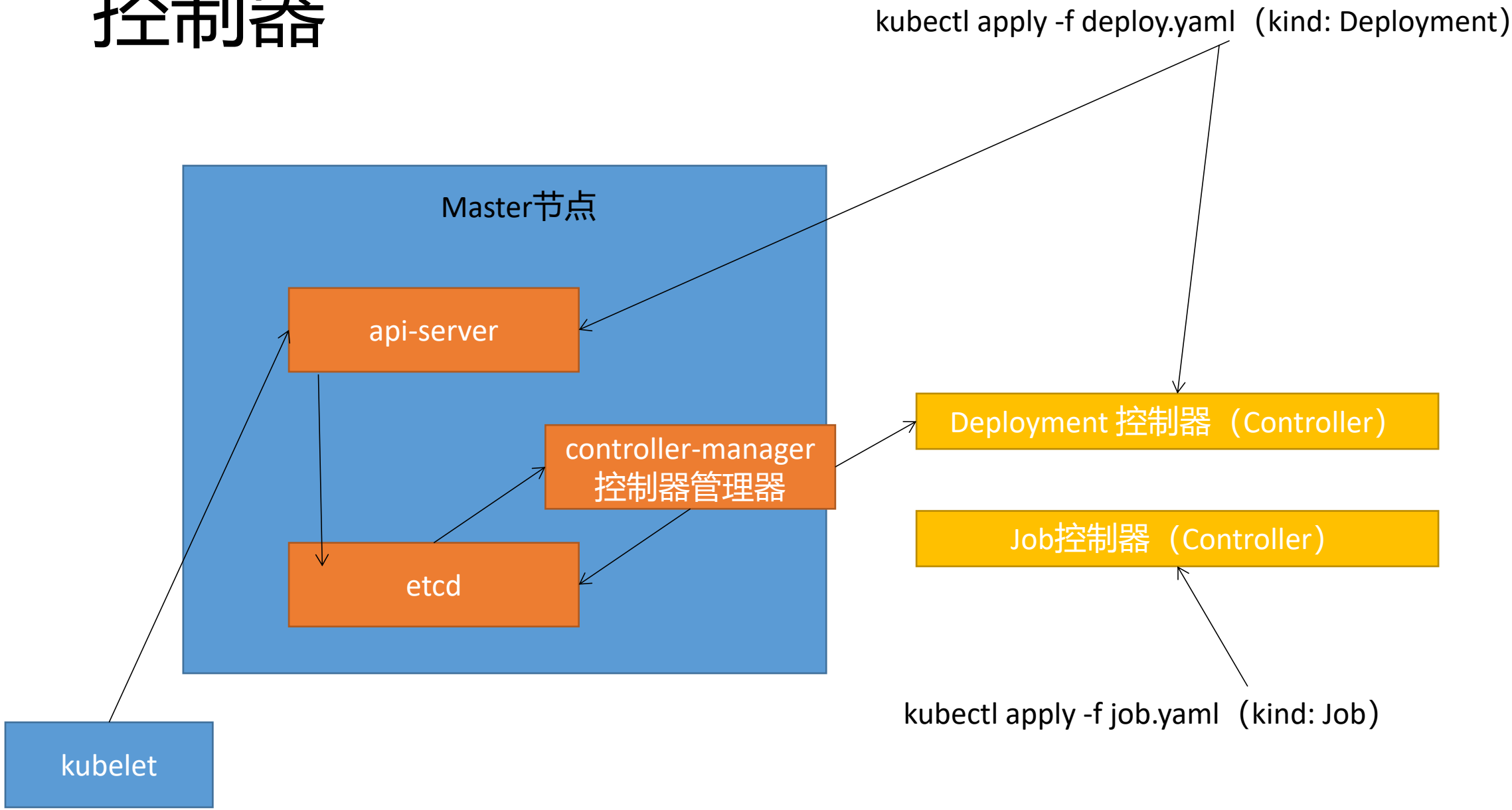
Pod


初始化容器组
init-container-0
init-container-1
init-container-N
Pod(容器组)
Container-0（应用容器）
Container-N（应用容器）
临时容器组
Ephemeral Container-0
Ephemeral Container-N

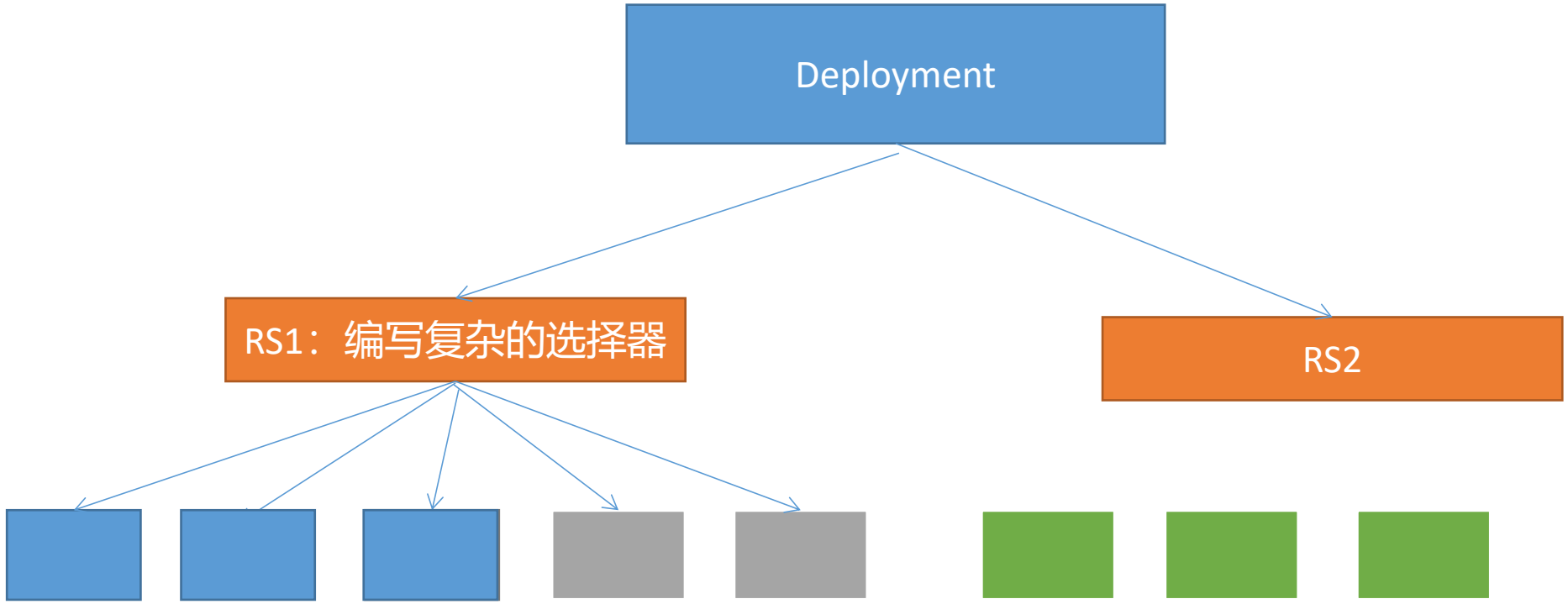
工作负载


Deploy
StatefulSet
DaemonSet
Job
CronJob
ReplicaSet
ReplicaController
Pod
container
Pod
container
container
Pod

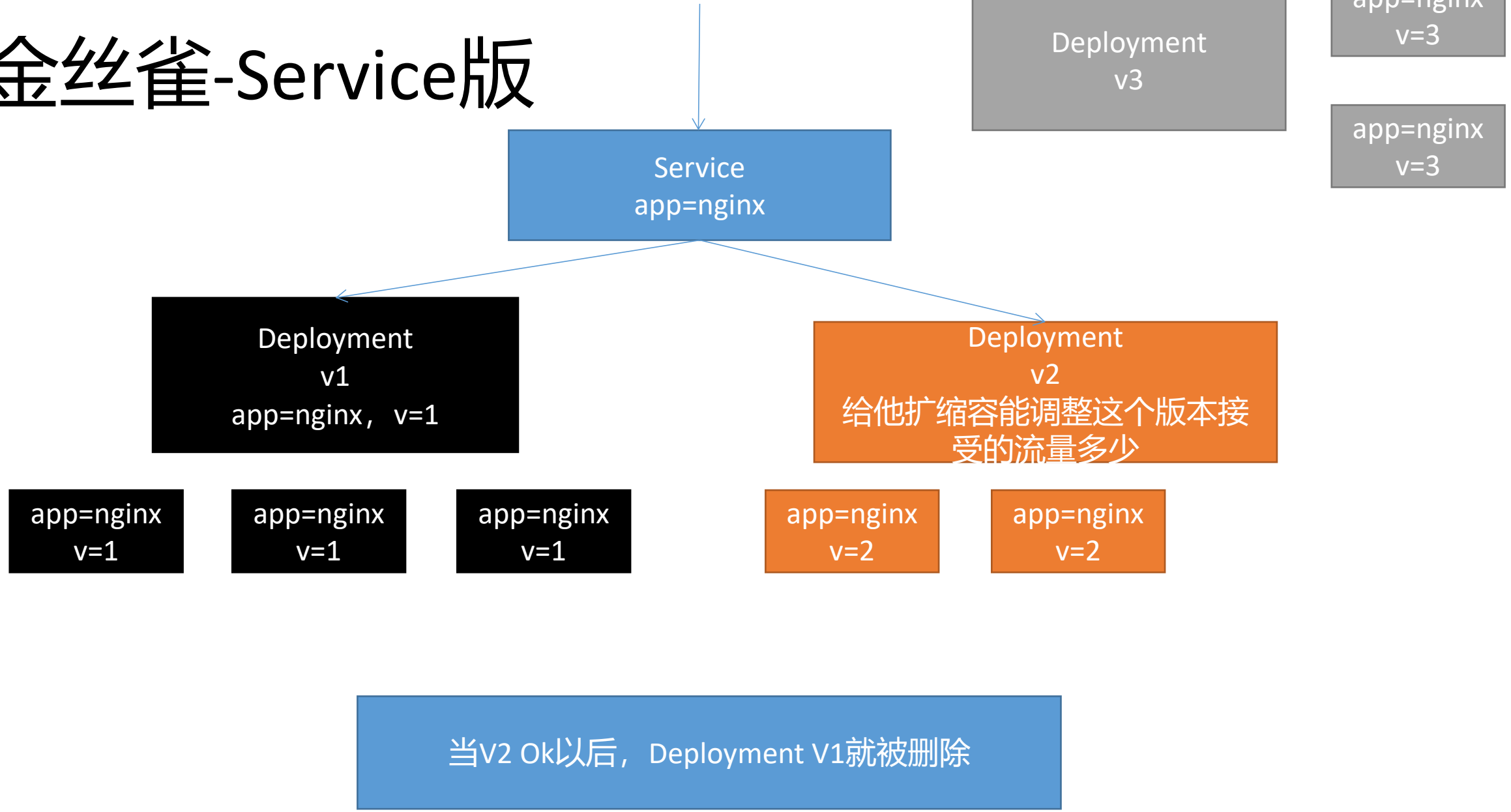
控制器


kubectl apply -f deploy.yaml （kind: Deployment）
Master节点
api-server
controller-manager
控制器管理器
etcd
Deployment 控制器（Controller）
Job控制器（Controller）
kubectl apply -f job.yaml （kind: Job）
kubelet

滚动更新

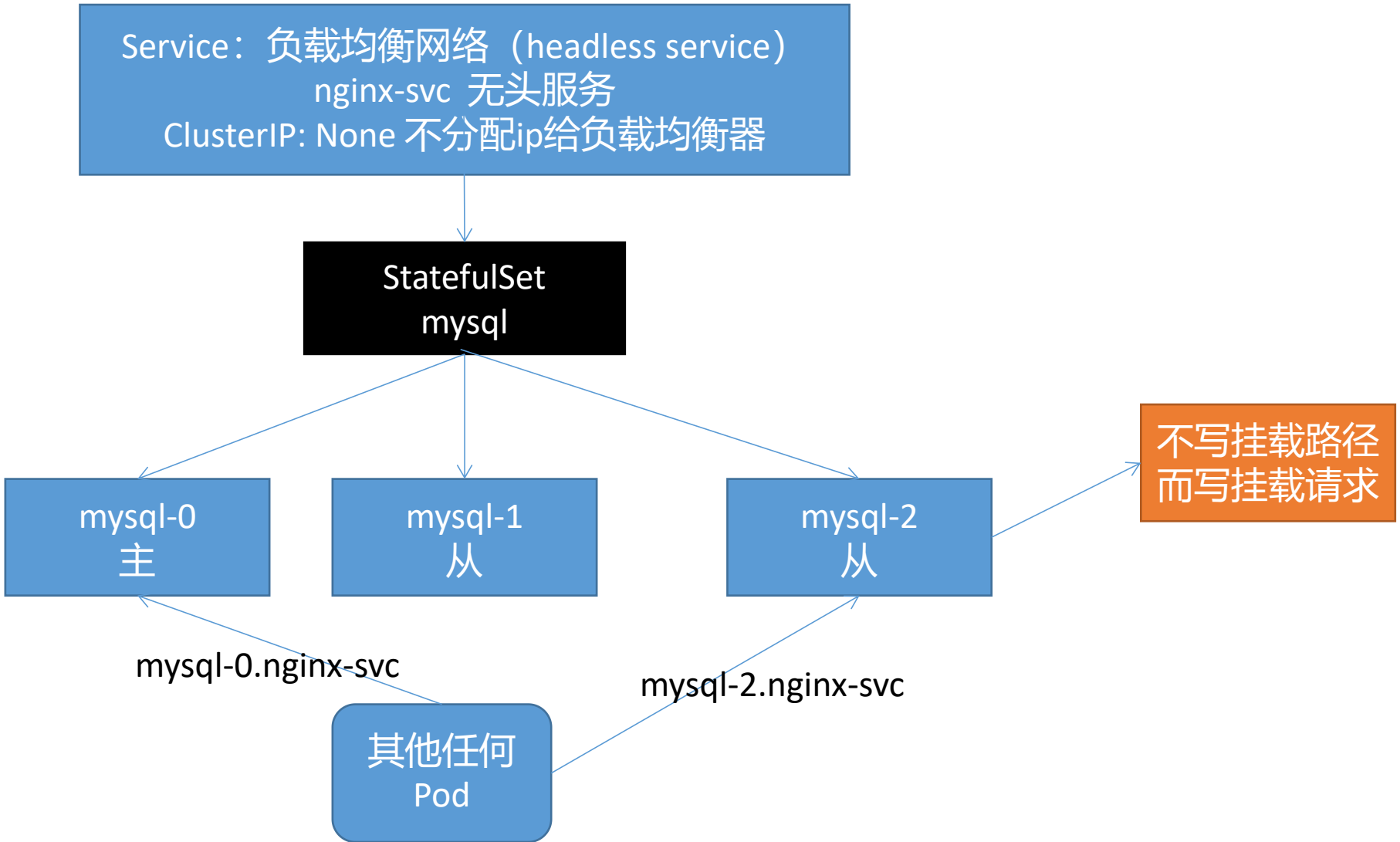


金丝雀-Service版



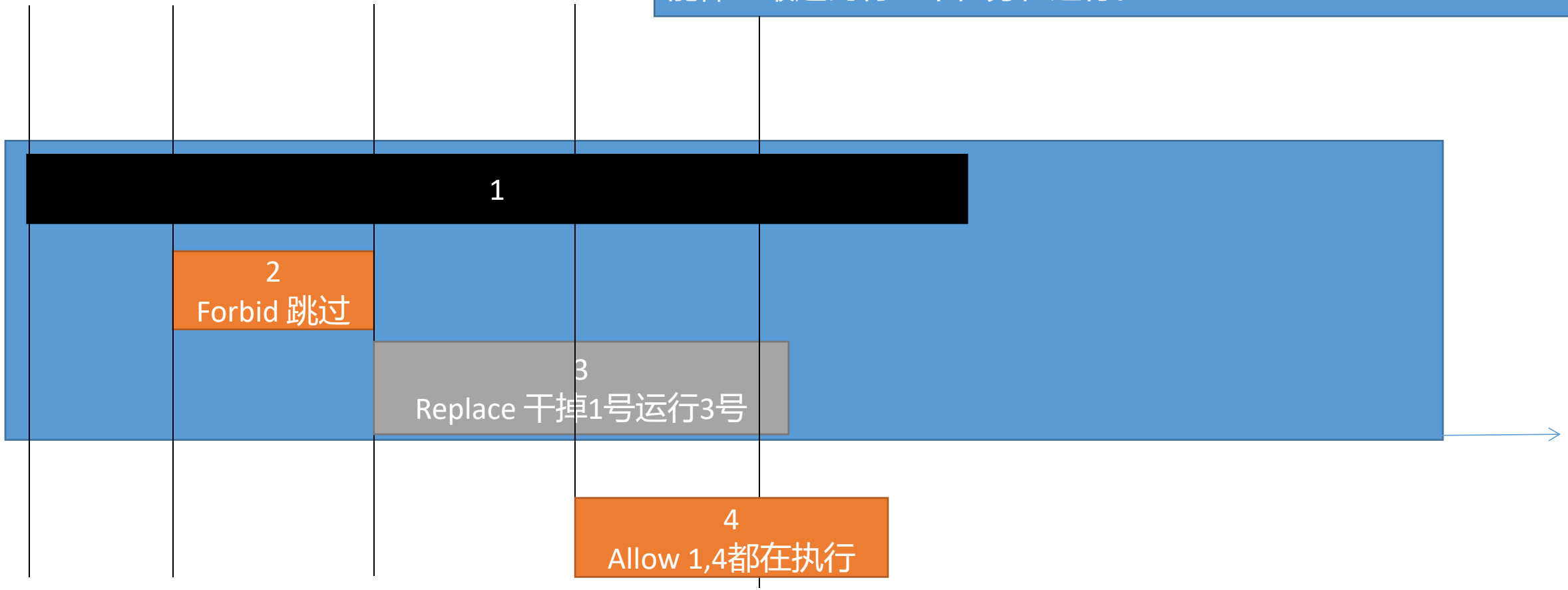
StatefulSet

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，defa
"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

containerPort: 8080

tomcat-container

containerPort: 6379

redis-container

targetPort: 8080

Pod

app: canary-tomcat

targetPort: 8080

Pod

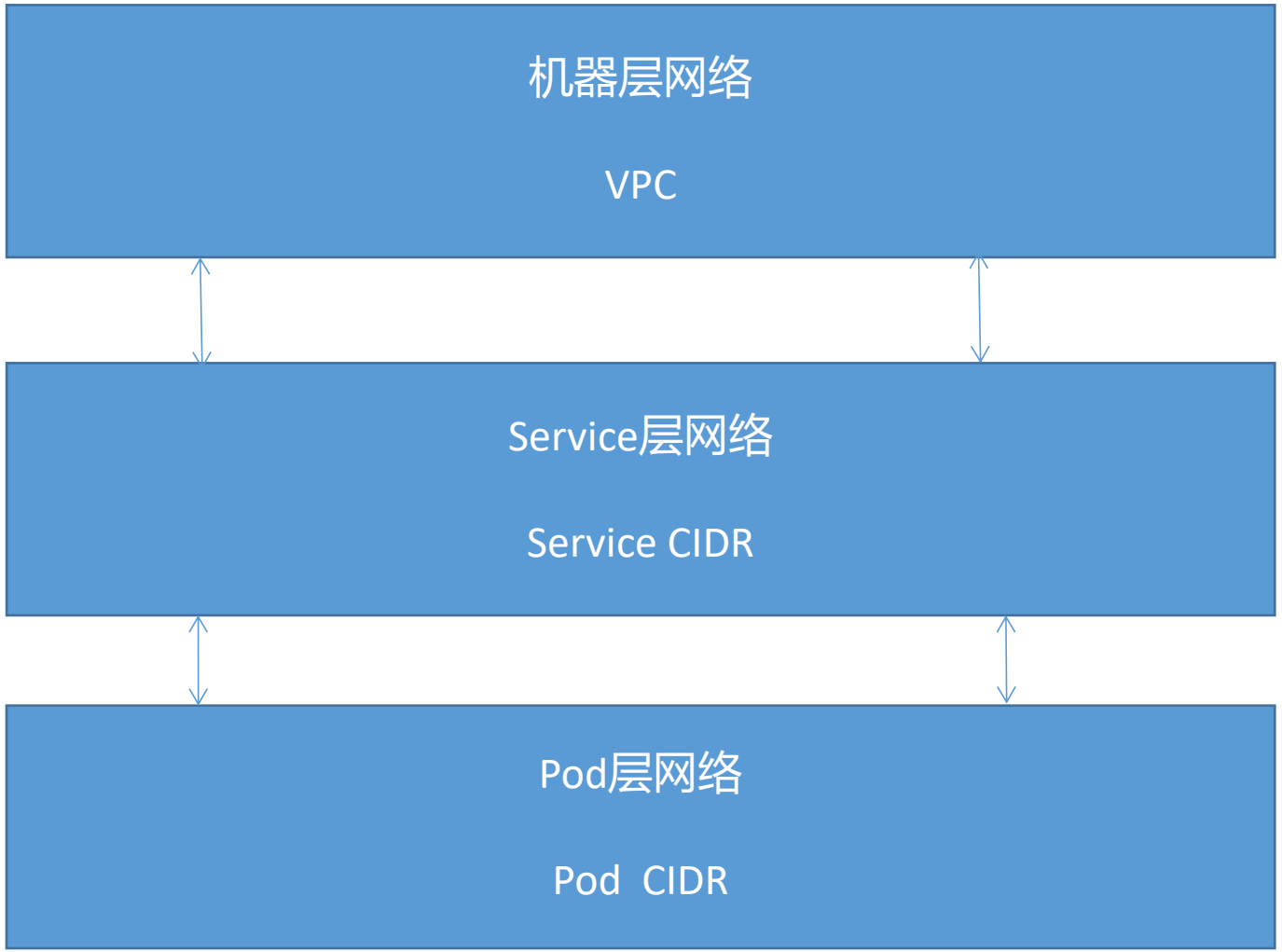
app: canary-tomcat

curl 10.170.11.11:80

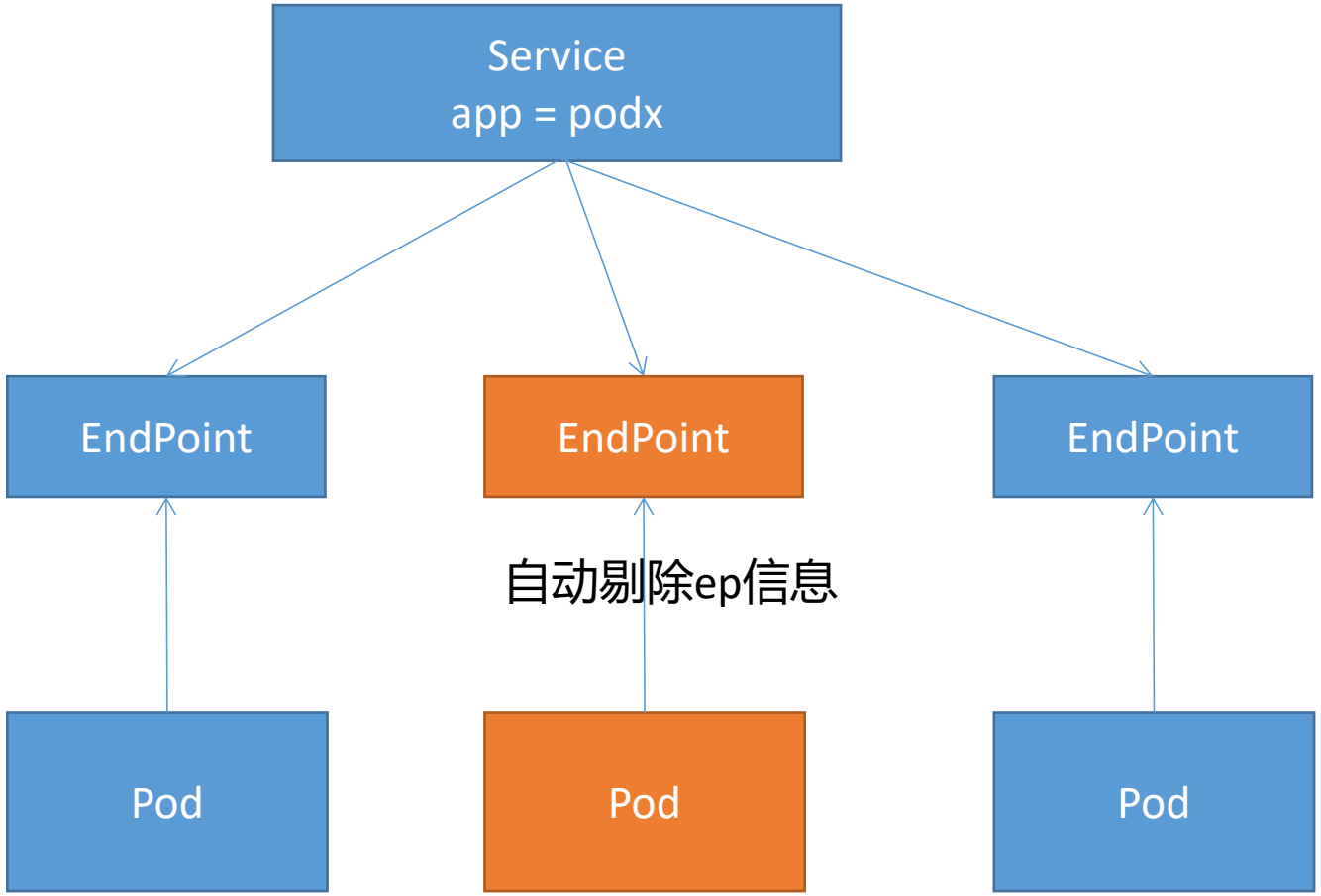
```
port: 80
targetPort: 8080
```

以上所有端口和Node的端口没有任何冲突

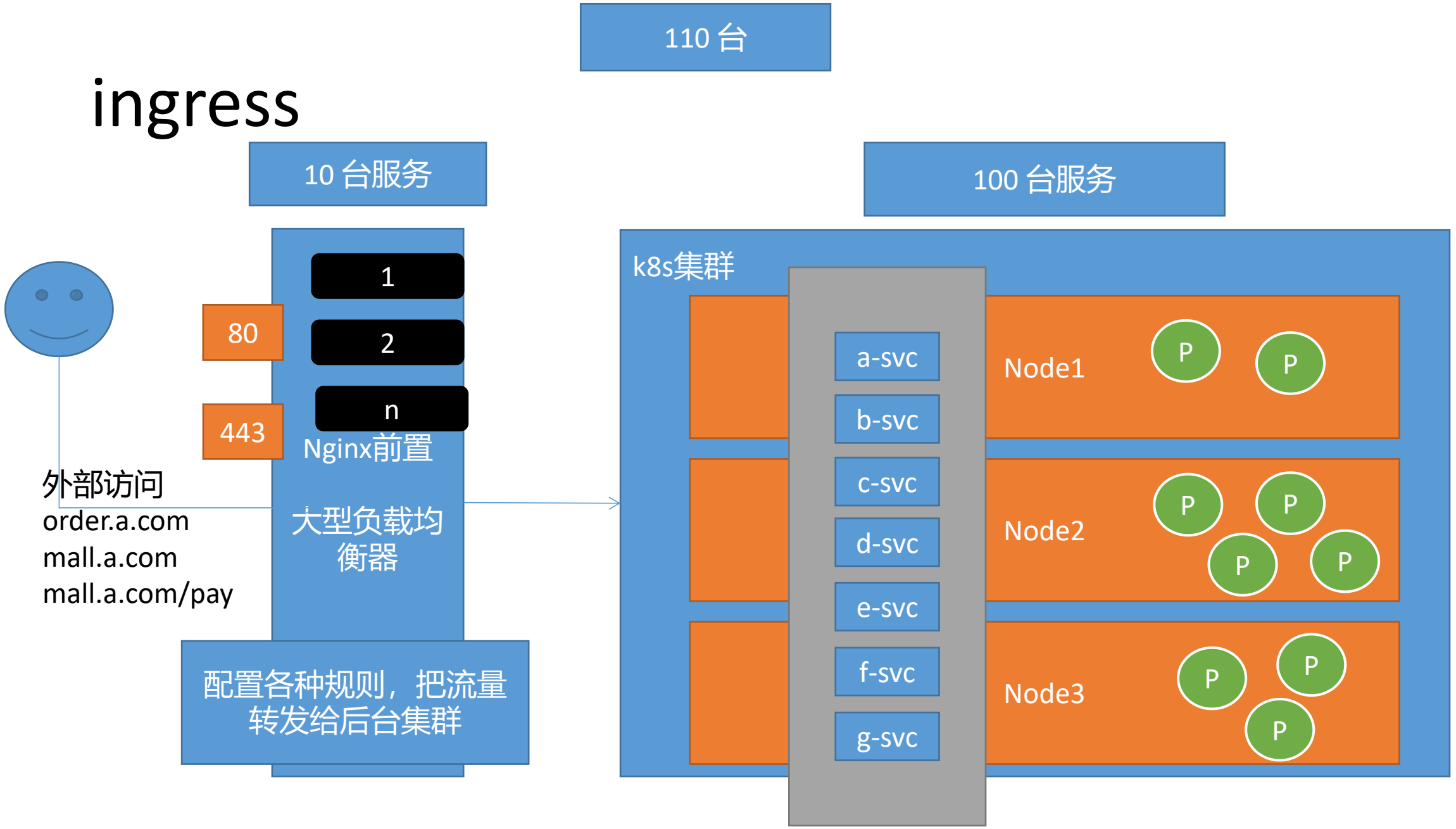
网络层次 -- 默认全是通的

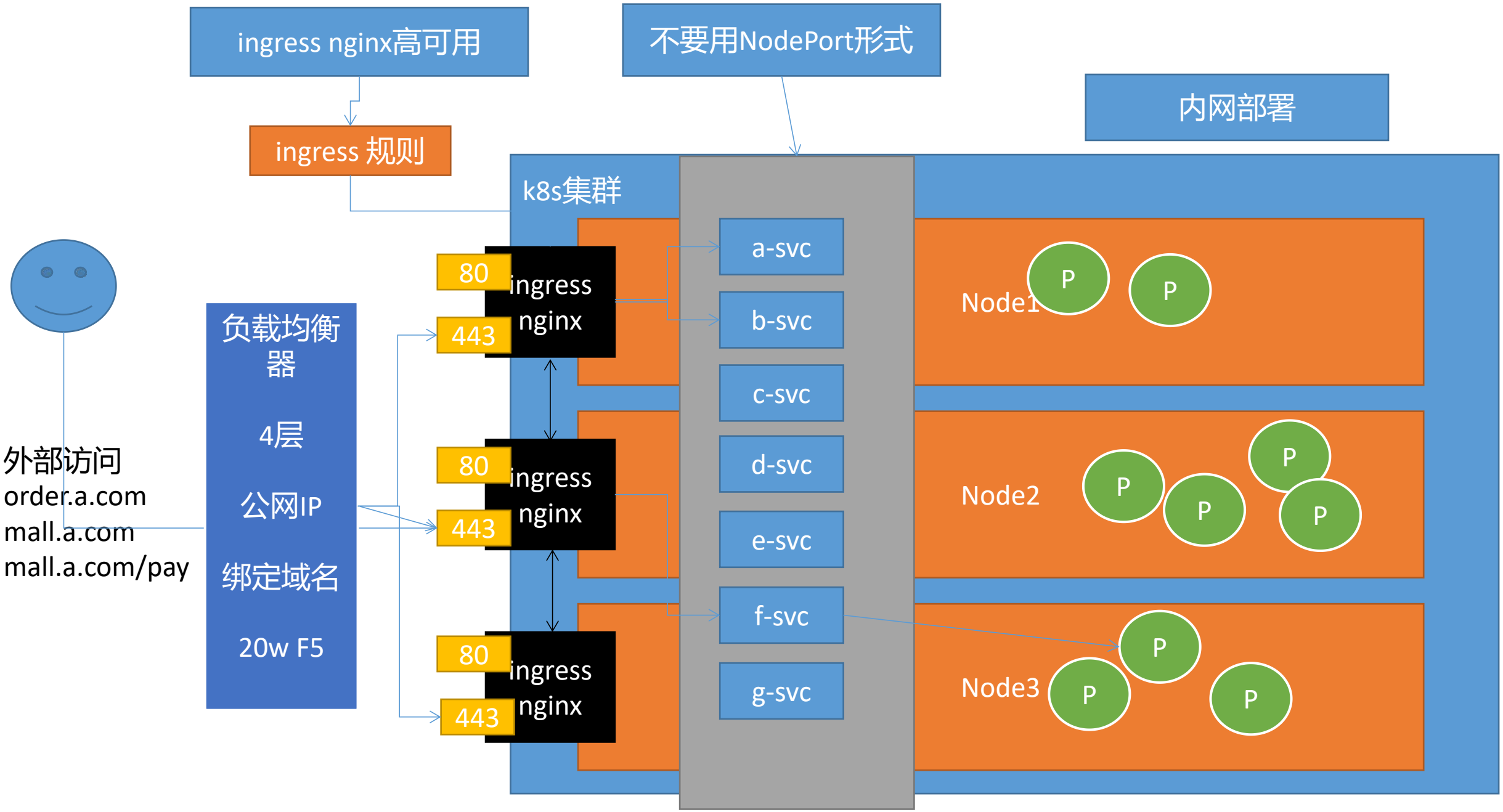


Service原理

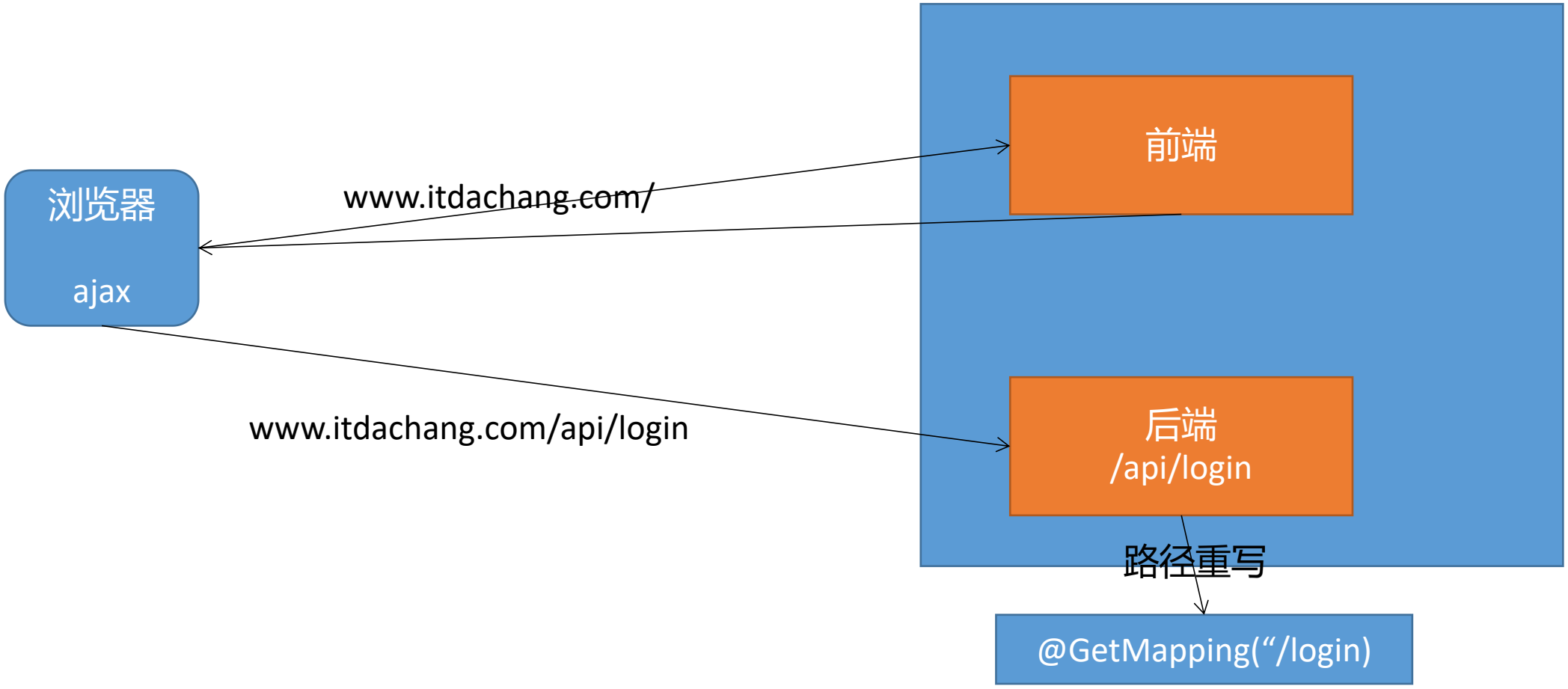

Service
app = podx
EndPoint
EndPoint
EndPoint
自动剔除ep信息
Pod
Pod
Pod

ingress

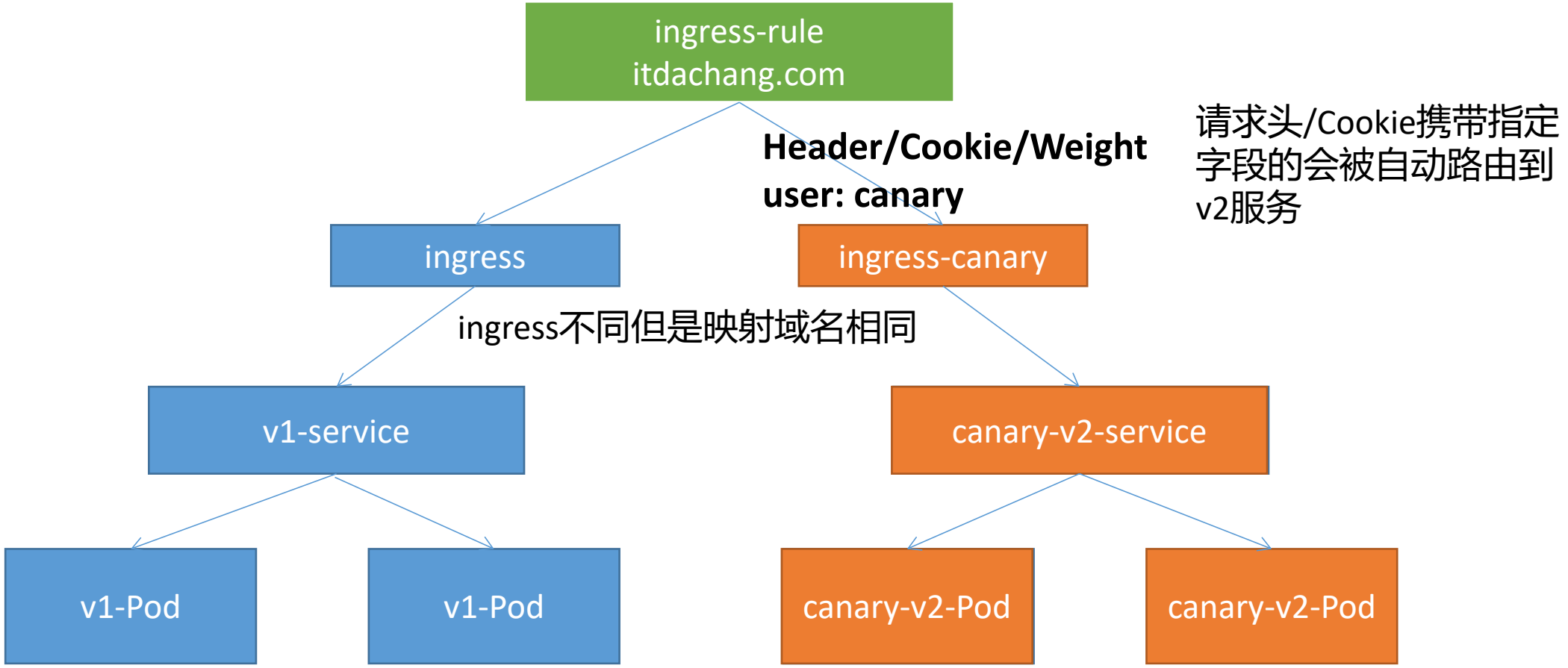

110 台
10 台服务
100 台服务
1
2
n
80
443
Nginx前置
大型负载均衡器
外部访问
order.a.com
mall.a.com
mall.a.com/pay
配置各种规则，把流量转发给后台集群
k8s集群
a-svc
b-svc
c-svc
d-svc
e-svc
f-svc
g-svc
Node1
Node2
Node3
P


ingress nginx高可用
不要用NodePort形式
内网部署
ingress 规则
k8s集群
外部访问
order.a.com
mall.a.com
mall.a.com/pay
负载均衡器
4层
公网IP
绑定域名
20w F5
80
443
ingress nginx
80
443
ingress nginx
80
443
ingress nginx
a-svc
b-svc
c-svc
d-svc
e-svc
f-svc
g-svc
Node1
P
P
Node2
P
P
P
P
Node3
P
P
P

路径重写

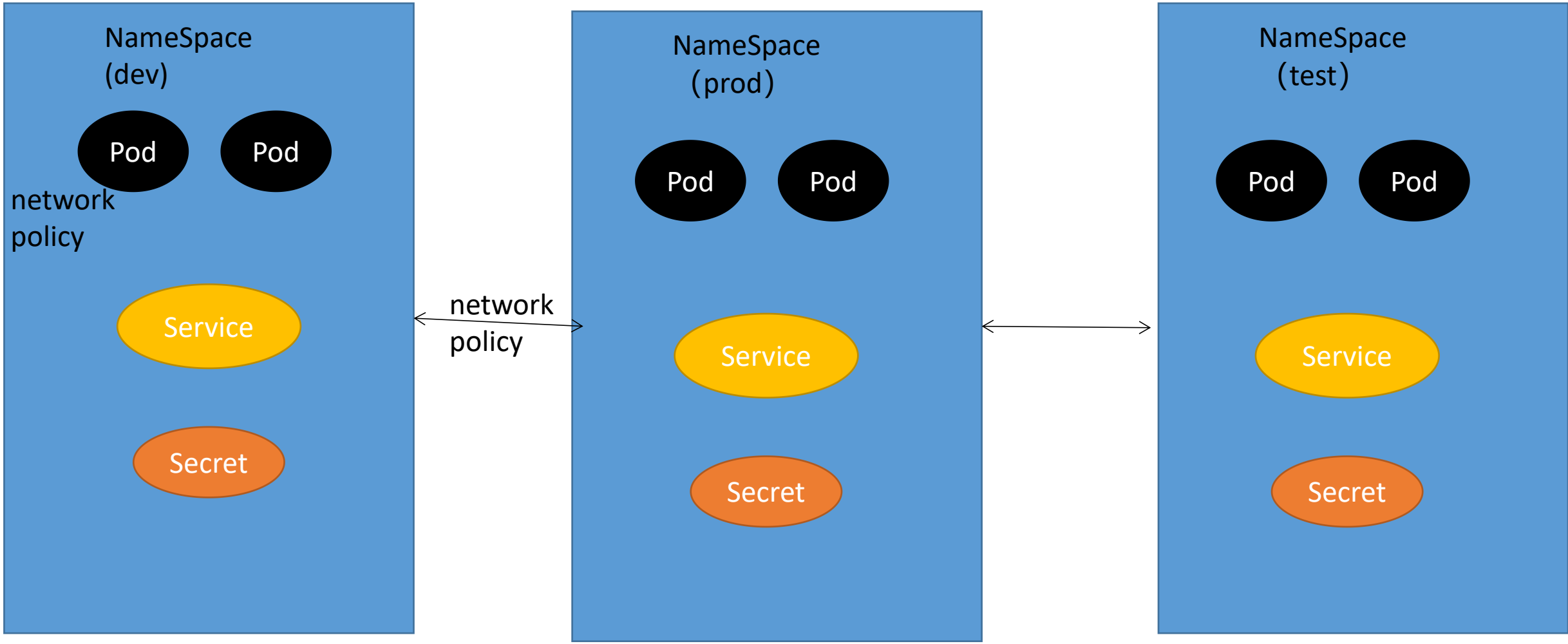

浏览器
ajax
www.itdachang.com/
www.itdachang.com/api/login
前端
后端
/api/login
路径重写
@GetMapping(“/login)

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

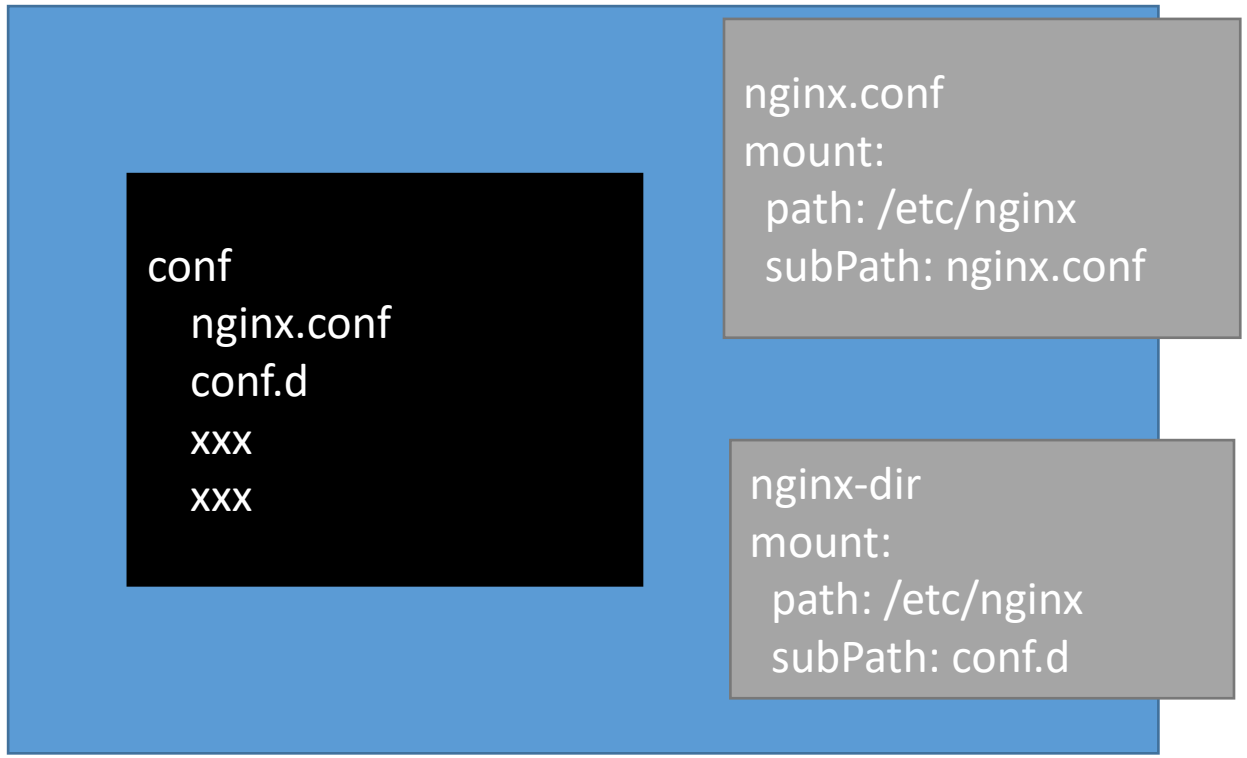
SpringBoot

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

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

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

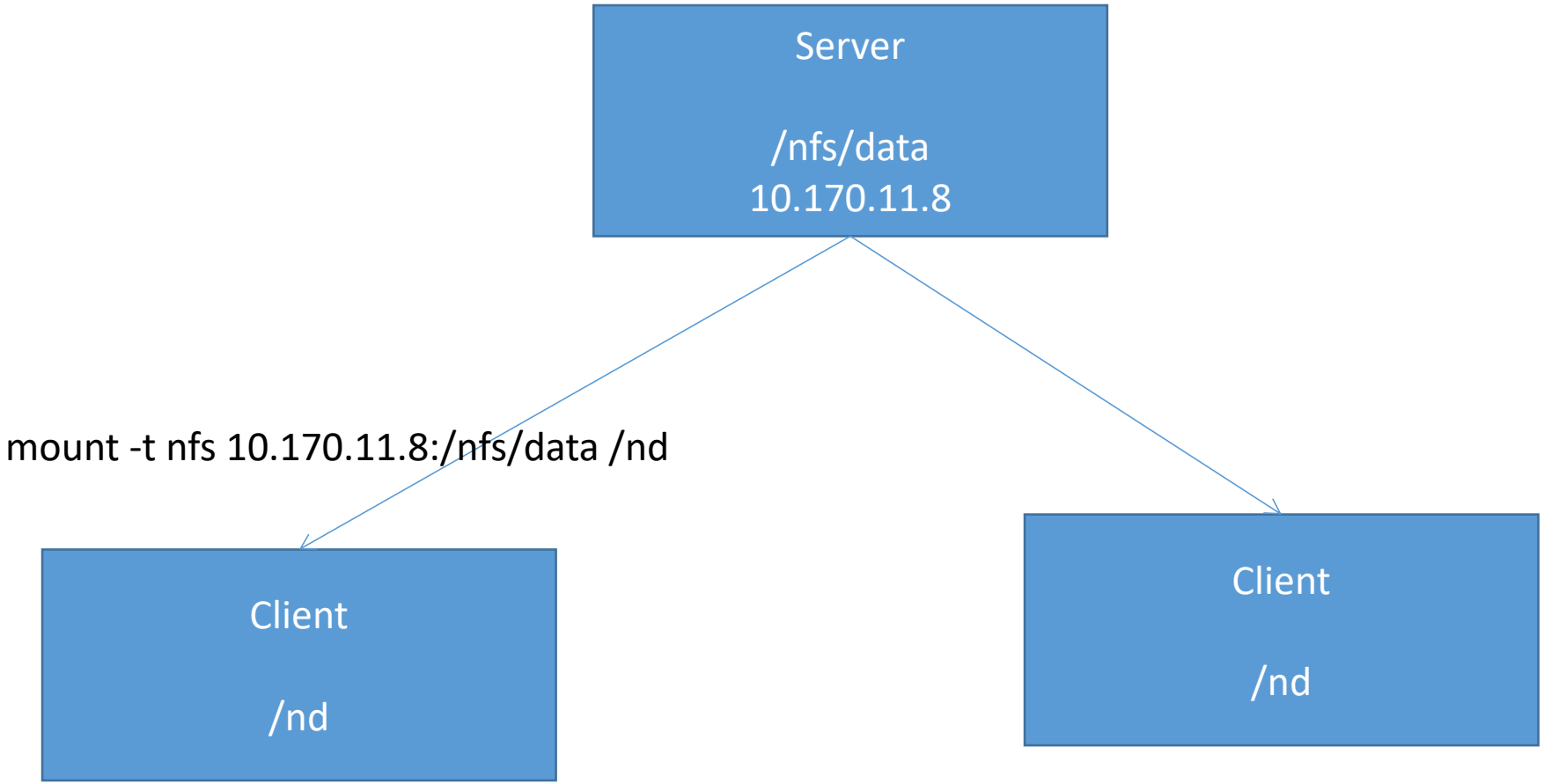
Nginx-可以使用子路径的方式



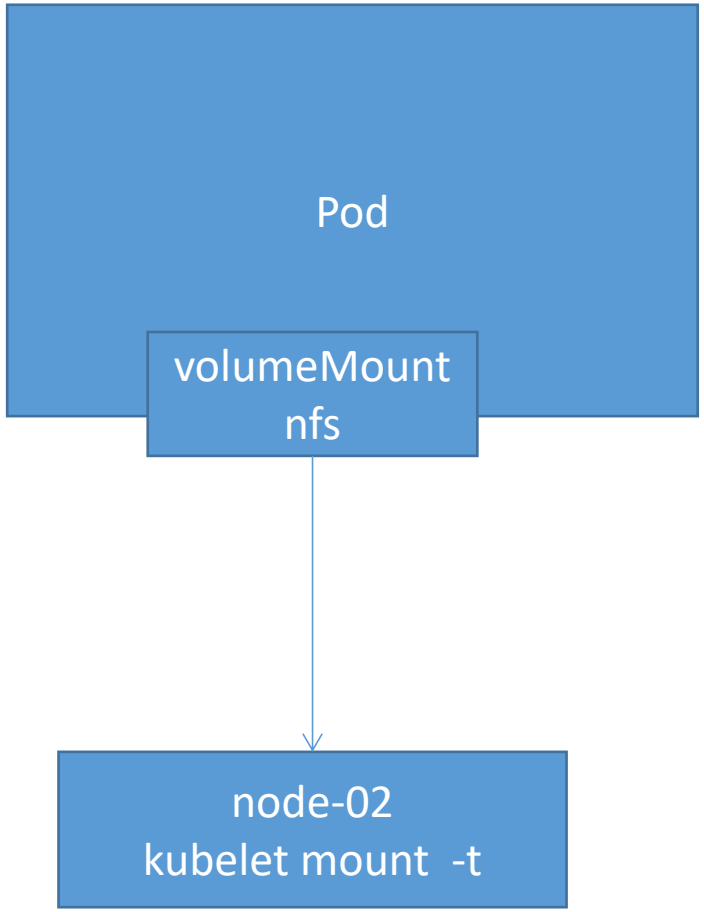

```
conf.d

 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

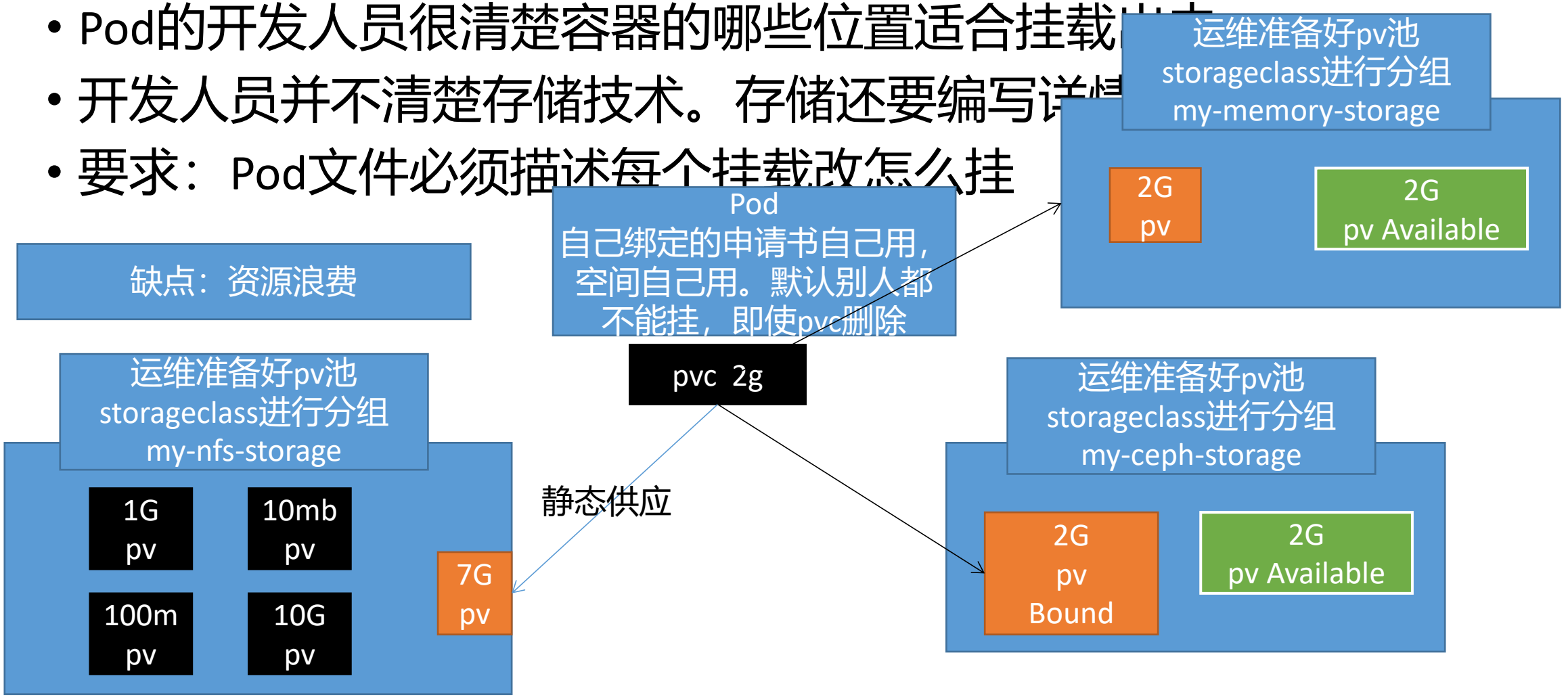

Server
/nfs/data
10.170.11.8
mount -t nfs 10.170.11.8:/nfs/data /nd
Client
/nd
Client
/nd

直接挂载

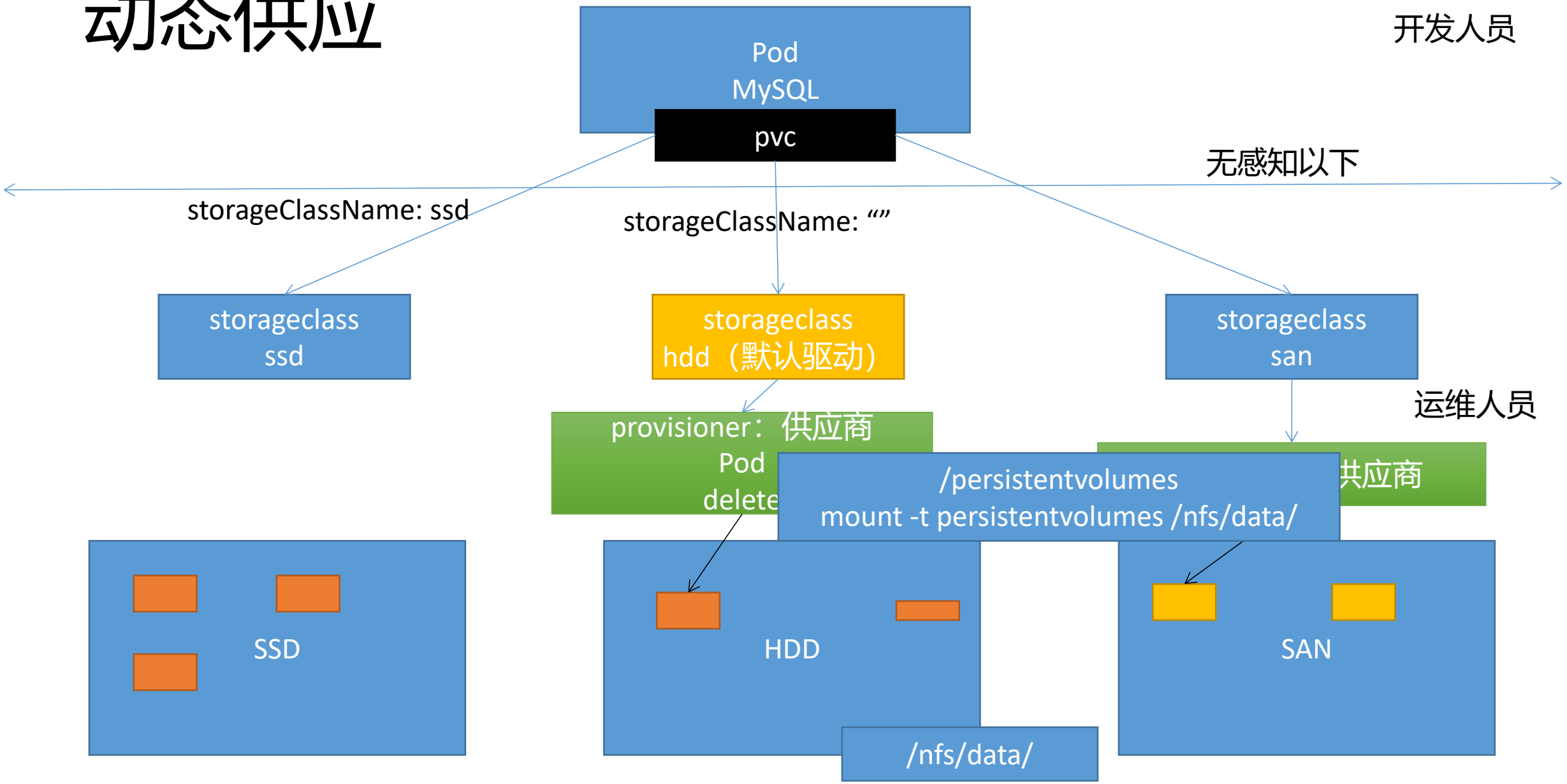


痛点

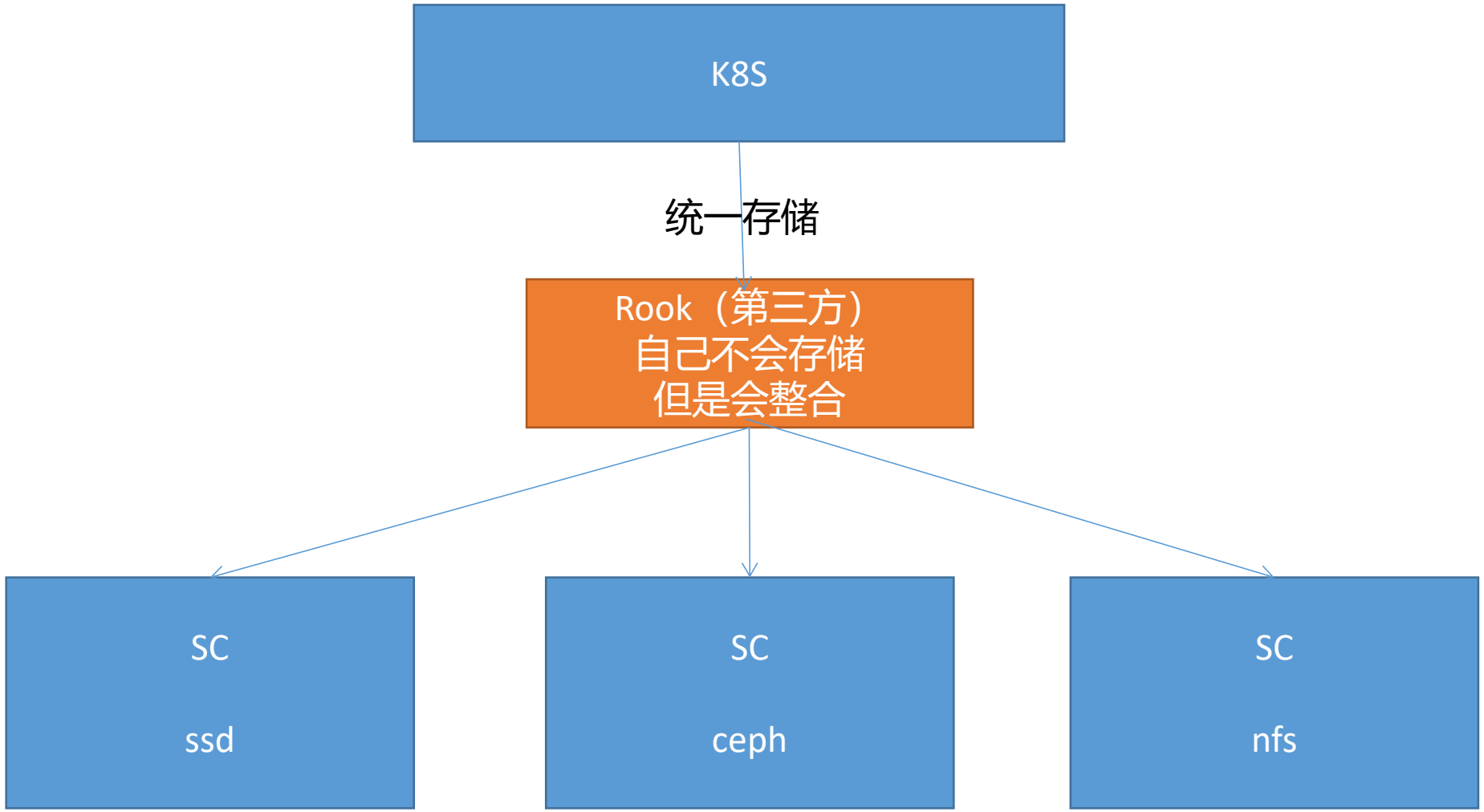
- Pod的开发人员很清楚容器的哪些位置适合挂载
- 开发人员并不清楚存储技术。存储还要编写详
- 要求：Pod文件必须描述每个挂载改怎么挂



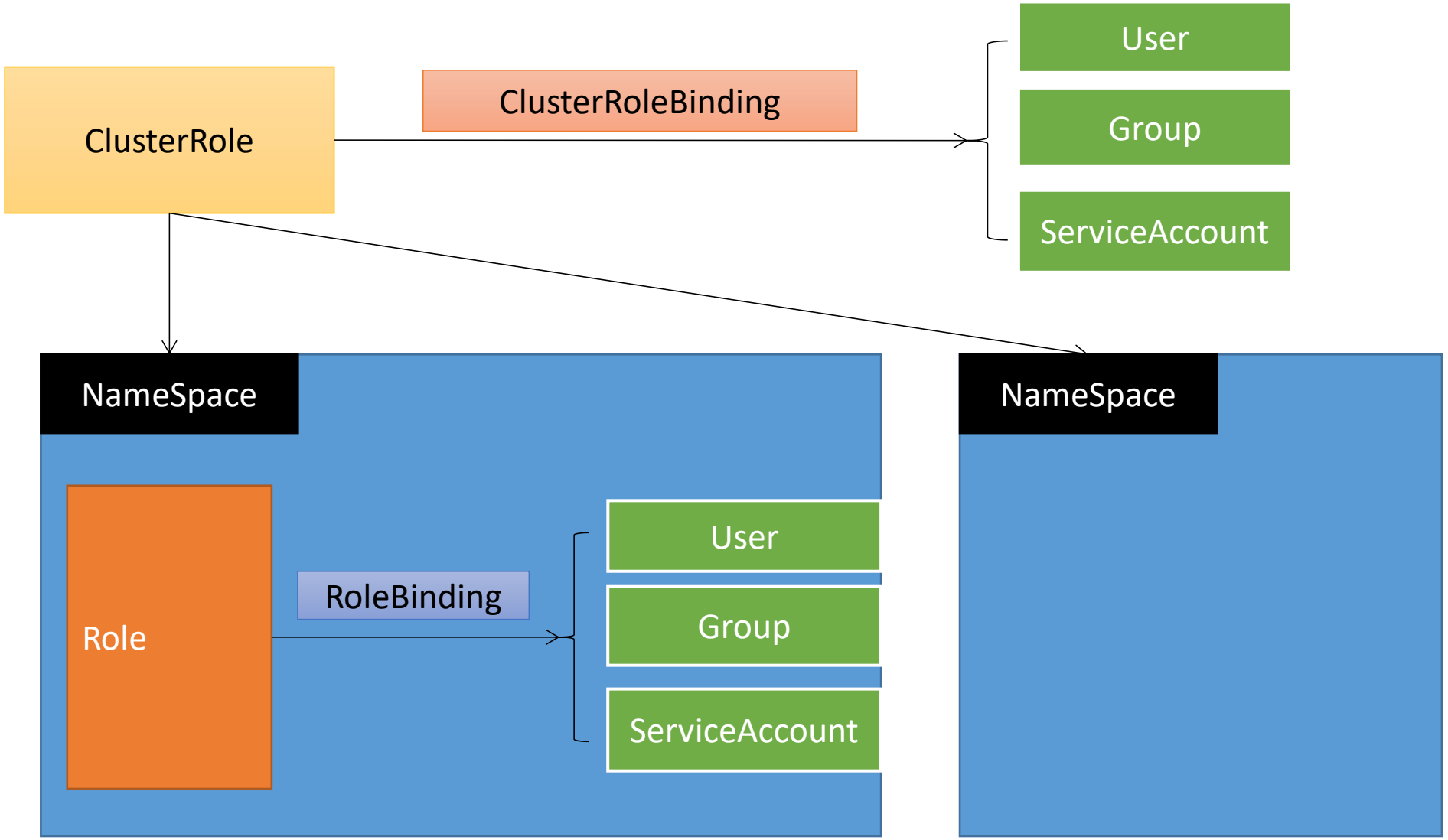
动态供应


开发人员
Pod
MySQL
pvc
无感知以下
storageClassName: ssd
storageClassName: “”
storageclass
ssd
storageclass
hdd（默认驱动）
storageclass
san
运维人员
provisioner：供应商
Pod
delete
/persistentvolumes
mount -t persistentvolumes /nfs/data/
SSD
HDD
SAN
/nfs/data/

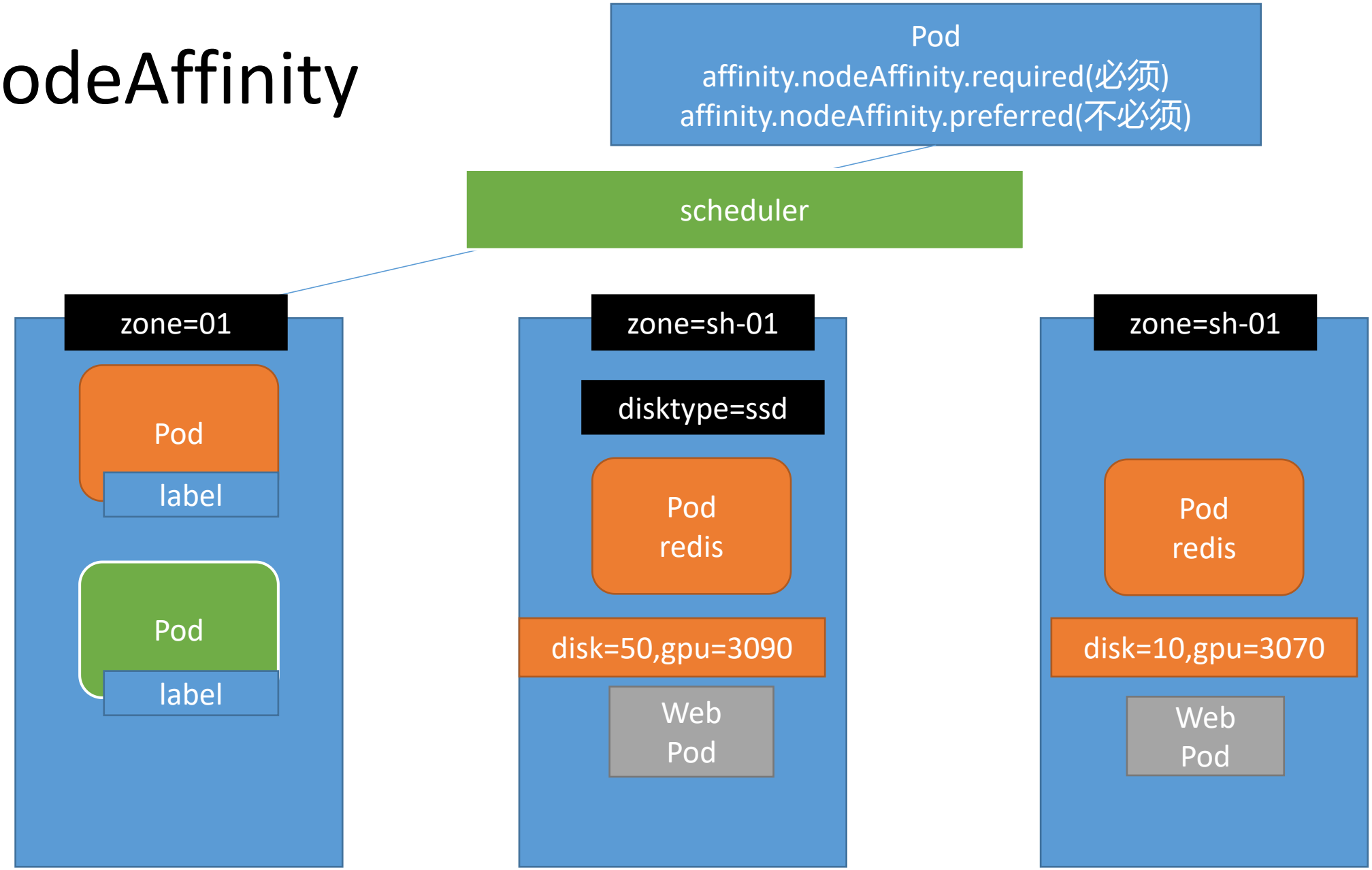
动态供应



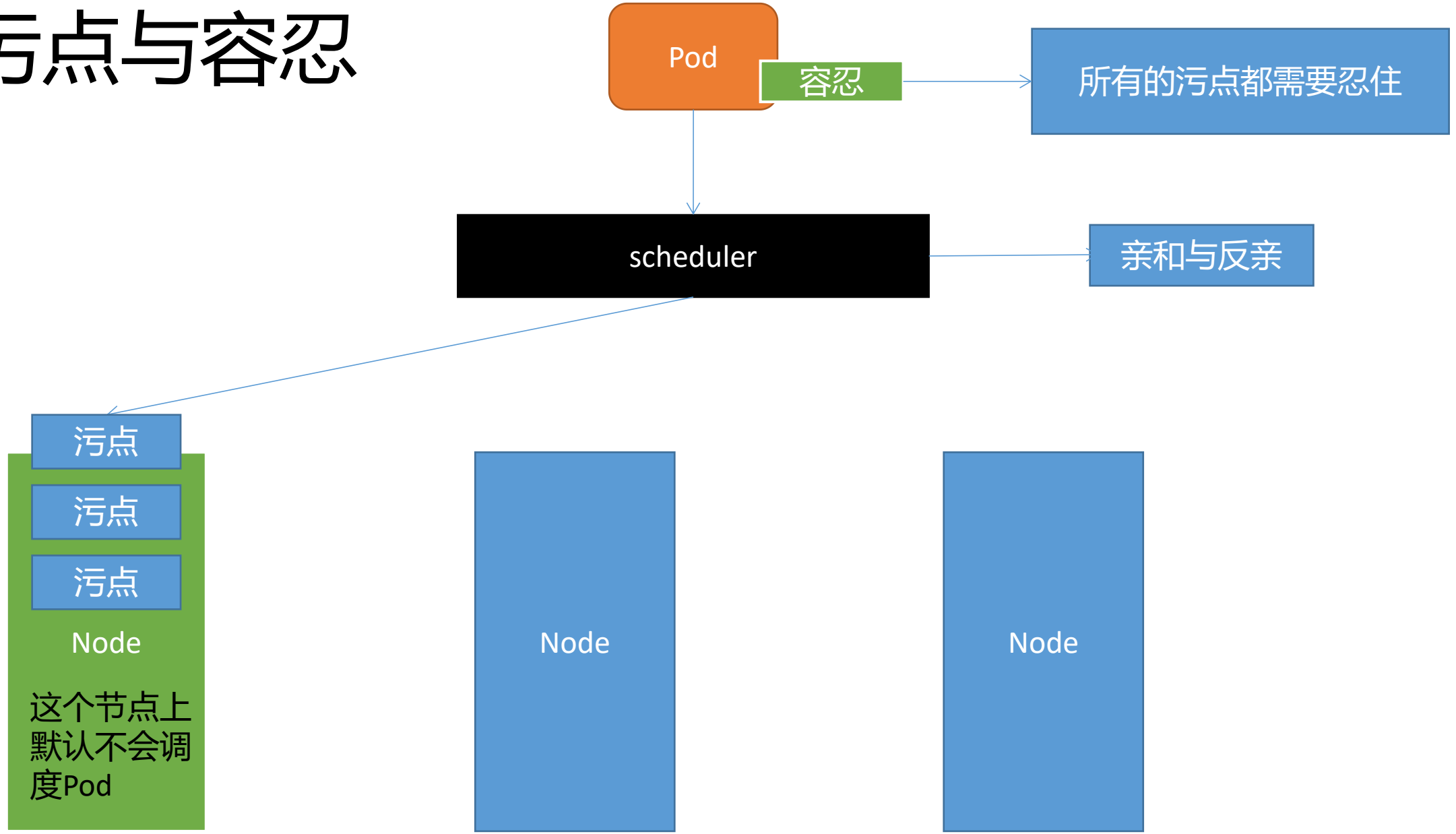
RBAC


ClusterRole
ClusterRoleBinding
User
Group
ServiceAccount
NameSpace
Role
RoleBinding
User
Group
ServiceAccount
NameSpace

nodeAffinity

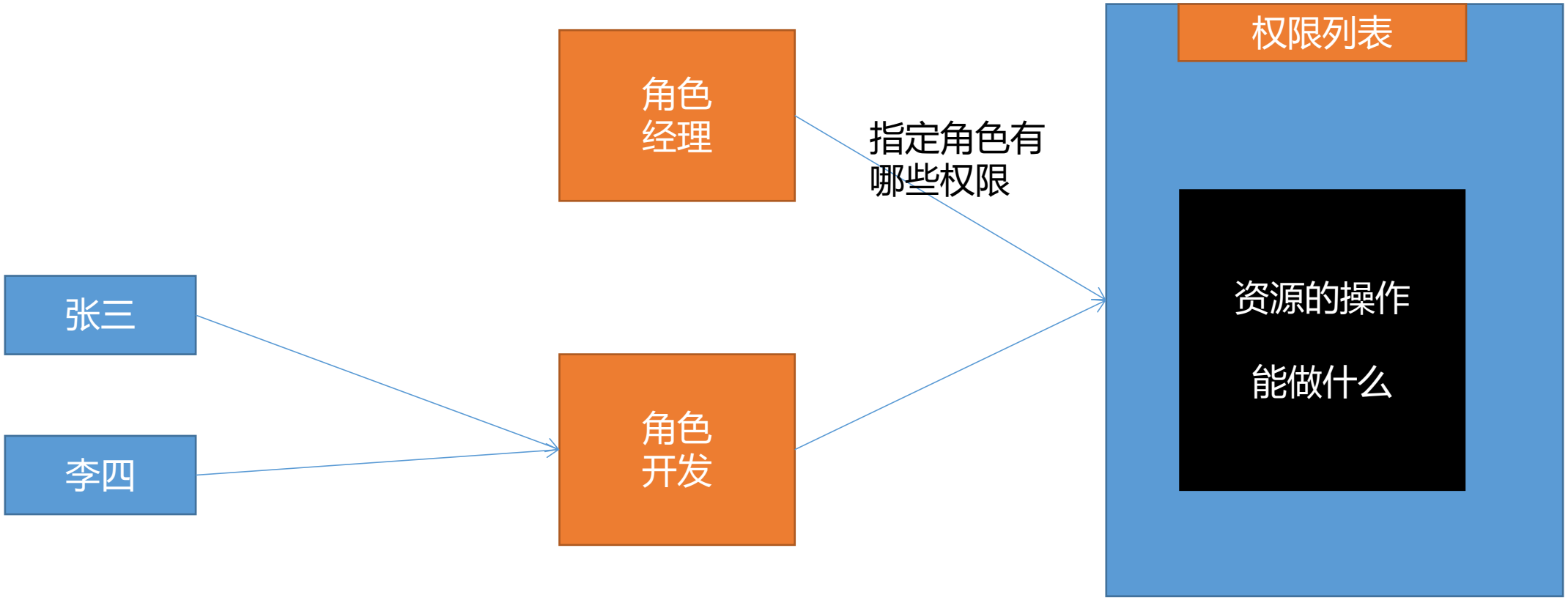


污点与容忍

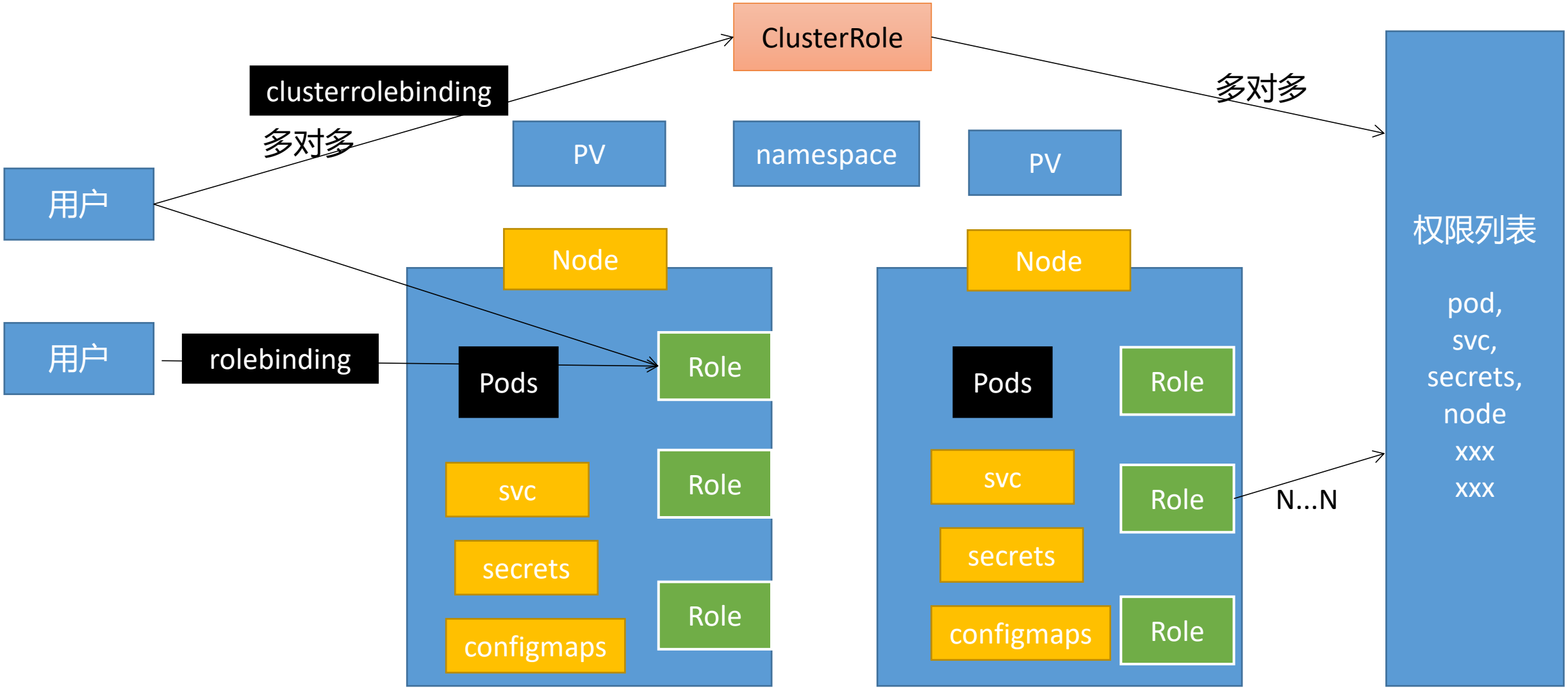

Pod
容忍
所有的污点都需要忍住
scheduler
亲和与反亲
污点
污点
污点
Node
这个节点上默认不会调度Pod
Node
Node

RBAC（Role-Based-Access-Controller）

- 基于角色的访问控制

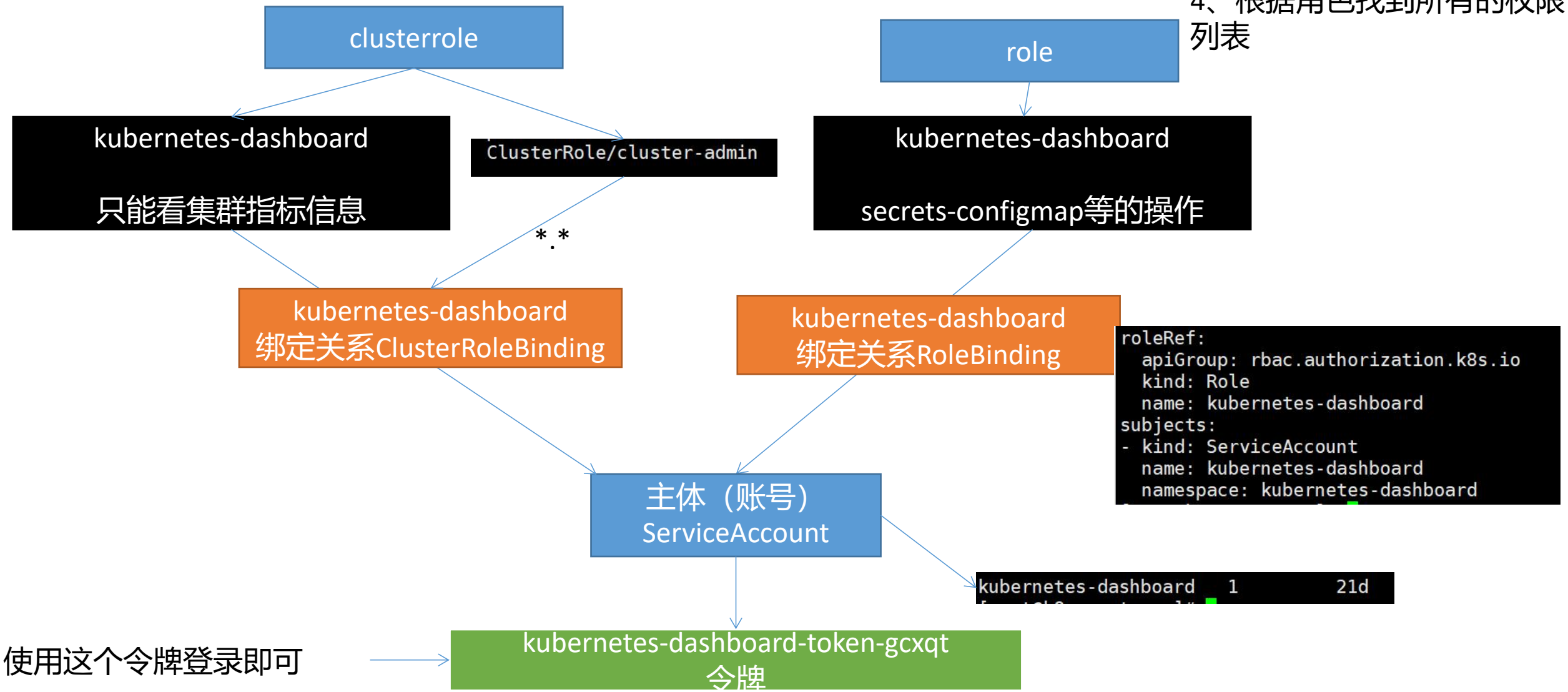


k8s-Role


ClusterRole
clusterrolebinding
多对多
多对多
PV
namespace
PV
用户
用户
rolebinding
Node
Pods
Role
svc
Role
secrets
Role
configmaps
Node
Pods
Role
svc
Role
secrets
configmaps
Role
N...N
权限列表
pod,
svc,
secrets,
node
xxx
xxx

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

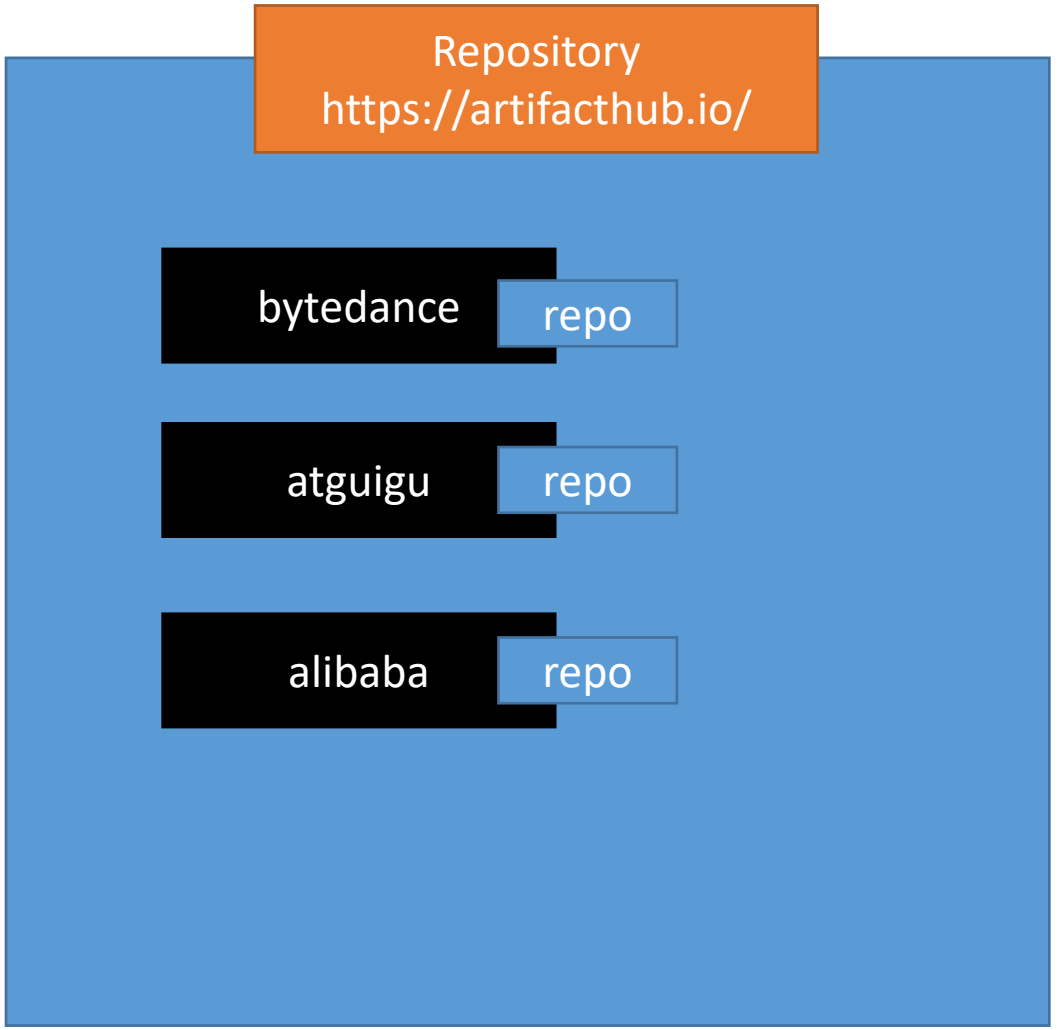


RBAC是这样

- 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
```