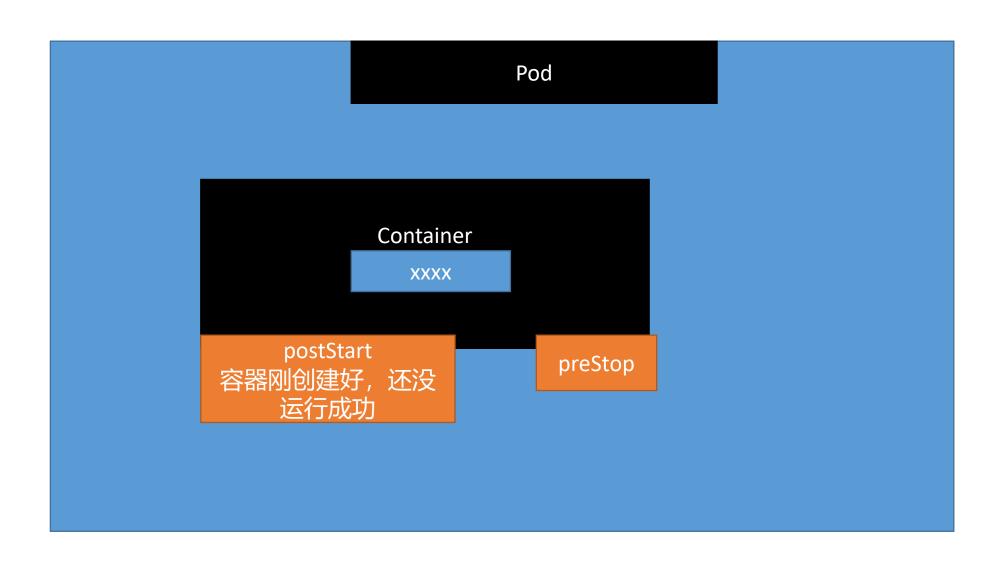
# K8S图例

### 容器

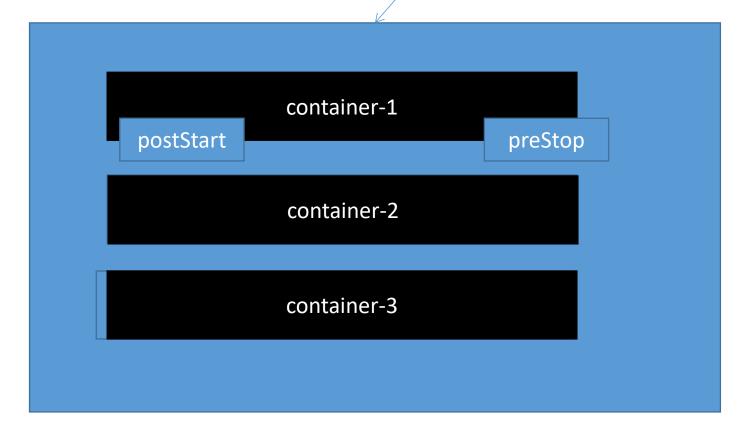


kubelet: 控制容器生命周期

### Pod

kubectl delete pod xxx

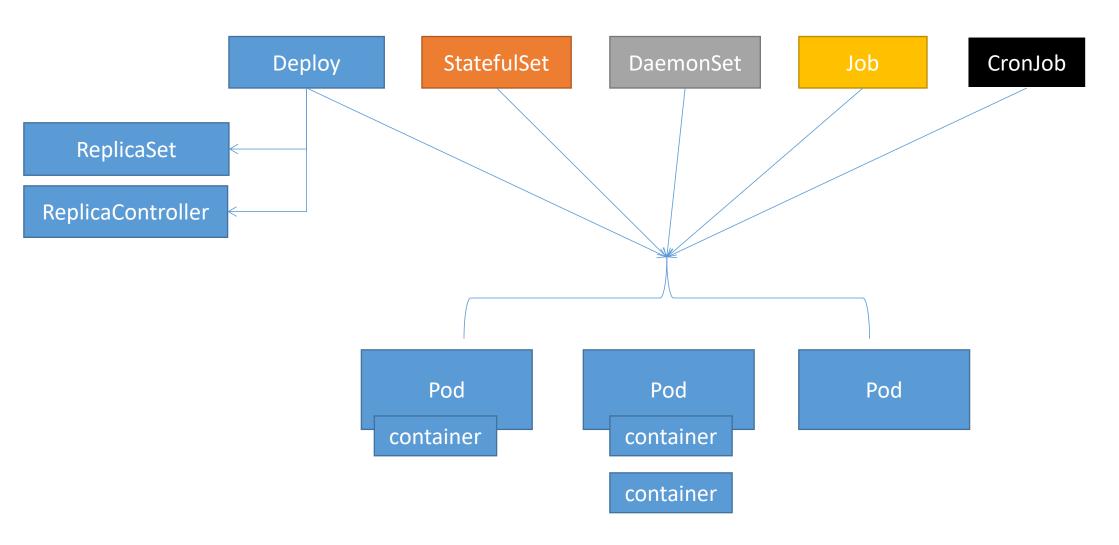
containers



#### Pod



# 工作负载



## 控制器

Master节点

controller-manager

控制器管理器

api-server

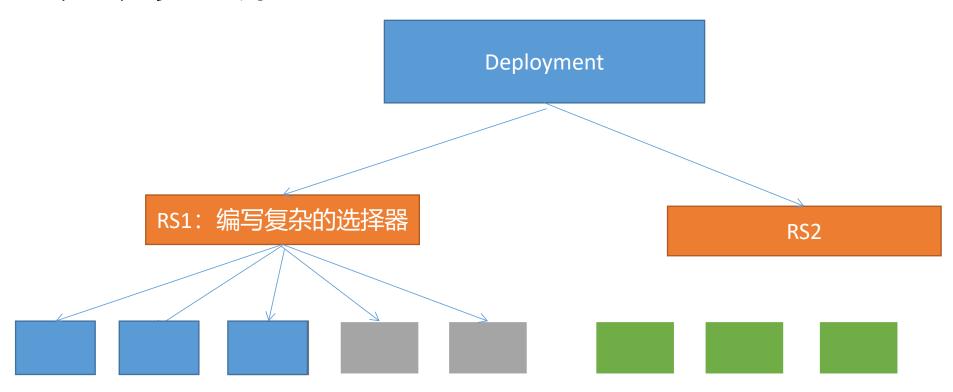
etcd

kubectl apply -f deploy.yaml (kind: Deployment) 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

Deployment v3

app=nginx v=3

app=nginx v=3

Service app=nginx

Deployment v1 app=nginx, v=1

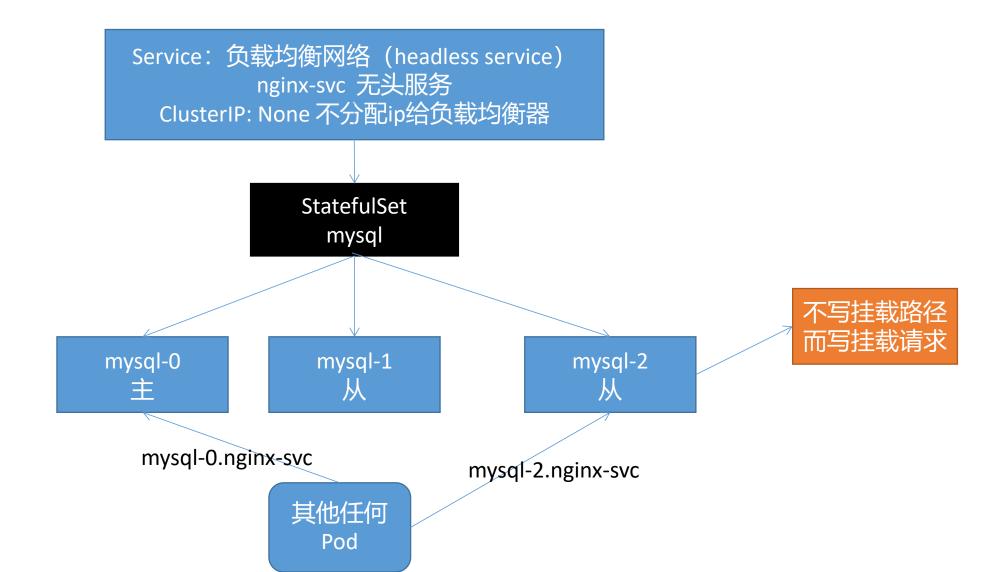
app=nginx v=1 app=nginx v=1 app=nginx v=1 Deployment v2 给他扩缩容能调整这个版本接 受的流量多少

app=nginx v=2 app=nginx v=2

当V2 Ok以后,Deployment V1就被删除

#### 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, defail "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<sup>curl 10.170.11.11:6379 访问不到</sup>

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

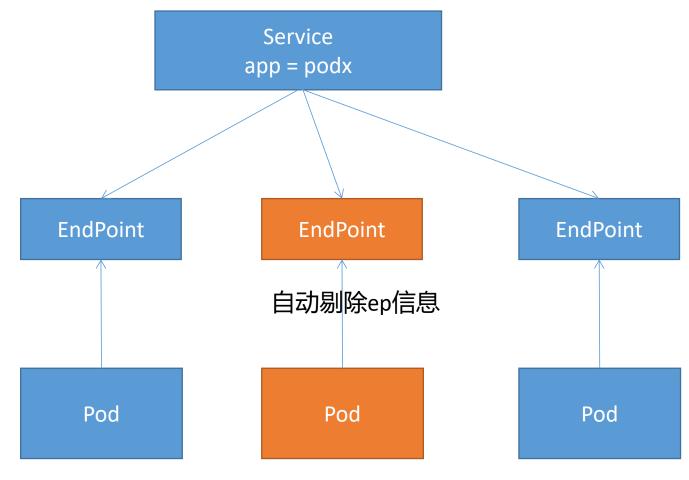
targetPort: 8080

以上所有端口和Node的端口没有任何冲突

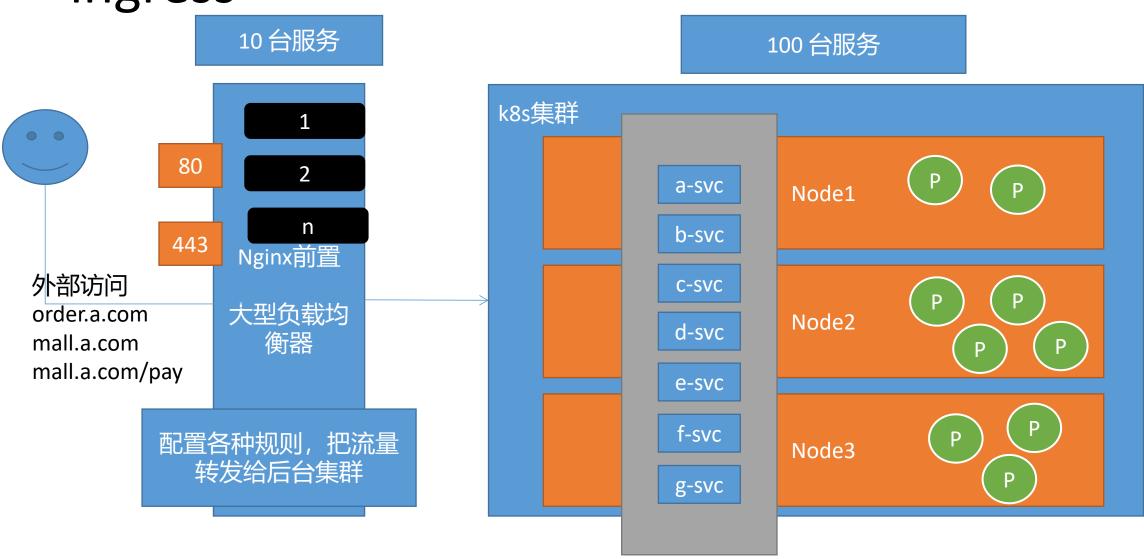
# 网络层次 -- 默认全是通的

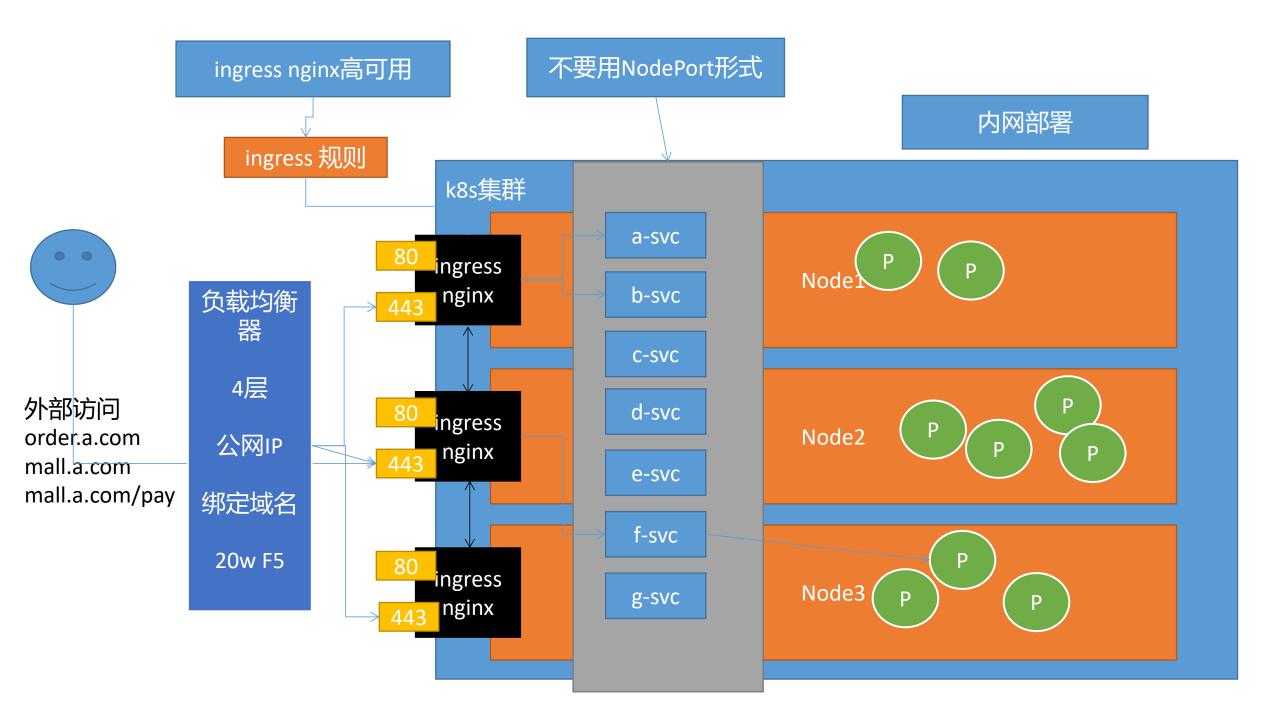


# Service原理

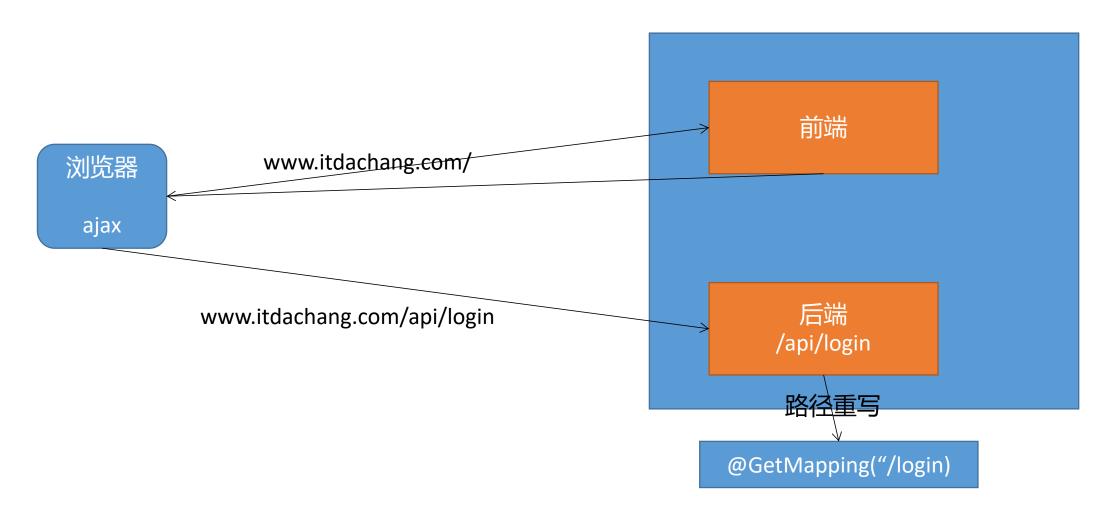


### ingress

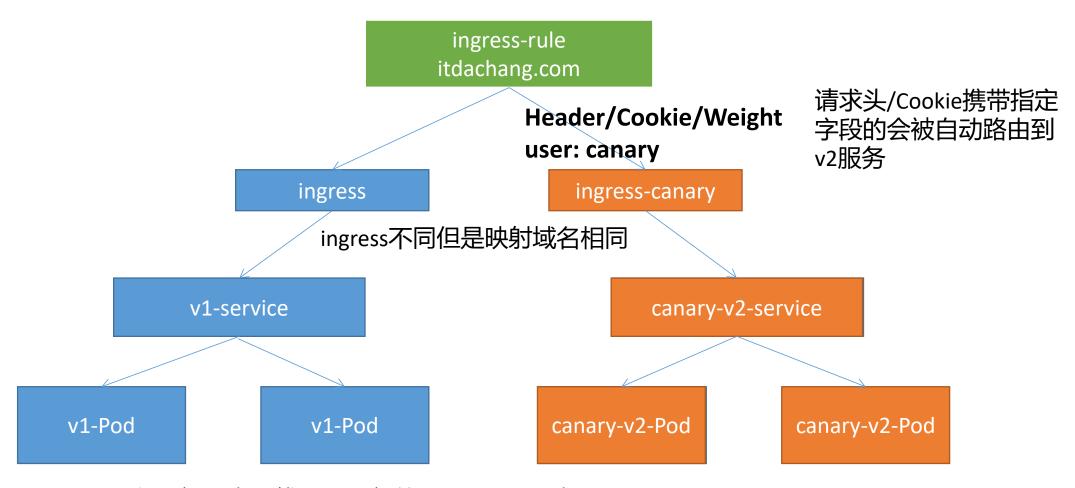




# 路径重写

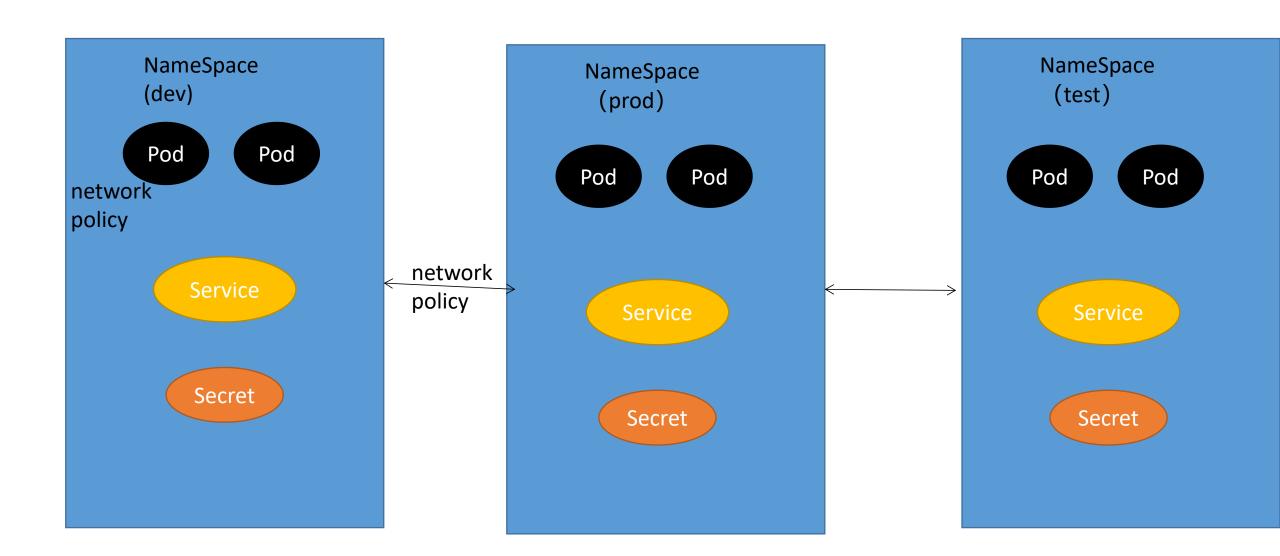


# 金丝雀-Ingress版



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 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

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

# Nginx-可以使用子路径的方式

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

nginx.conf

conf.d
xxx
xxx

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

#### conf.d

#### volumes:

- name: nginx-conf

hostPath:

path: /app/nginx/nginx.conf

type: File

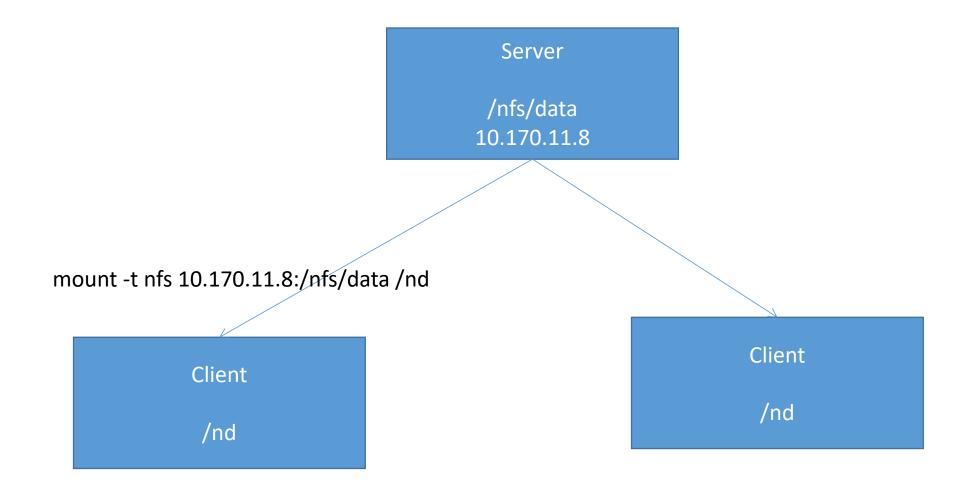
- name: nginx-dir

hostPath: ## 主机的这个文件

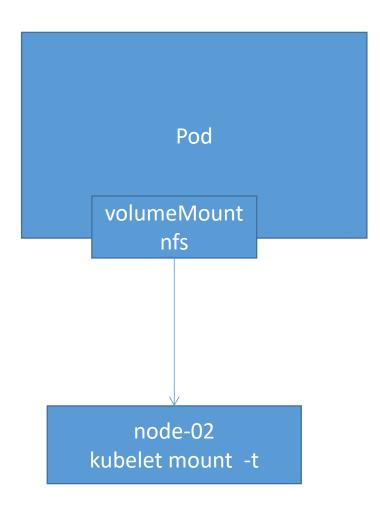
path: /app/nginx/conf.d

type: Directory

### NFS



# 直接挂载



### 痛点

100m

pv

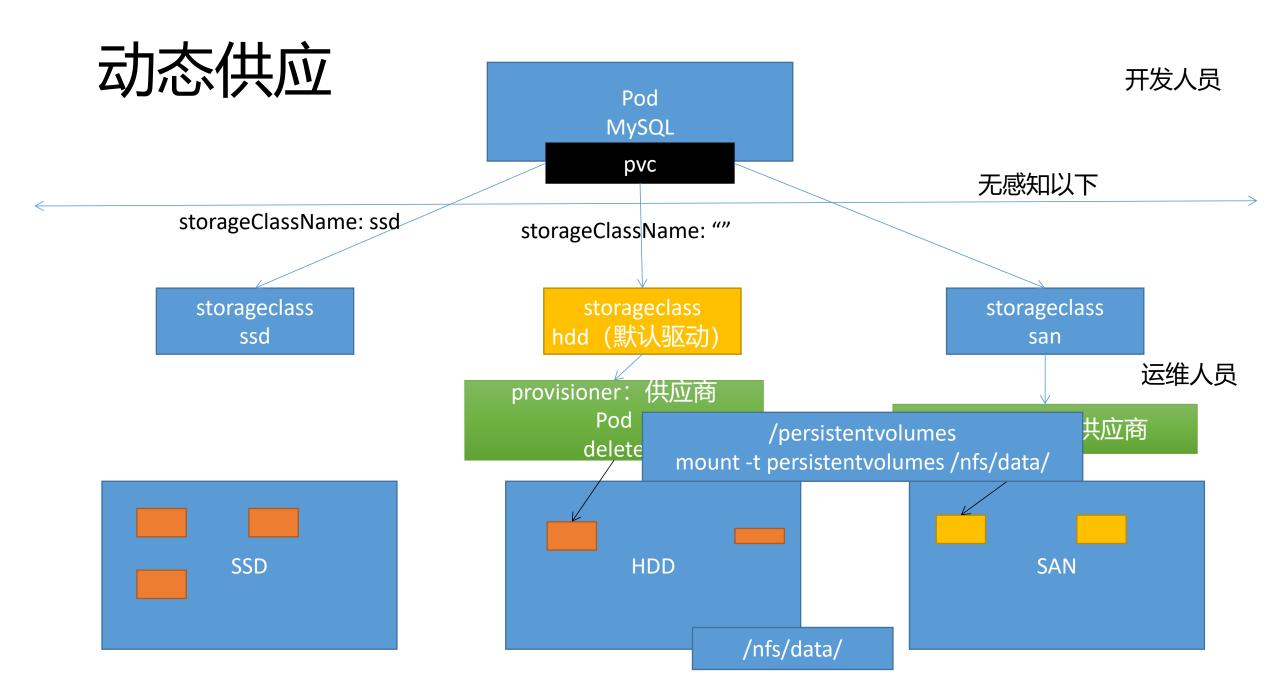
10G

pν

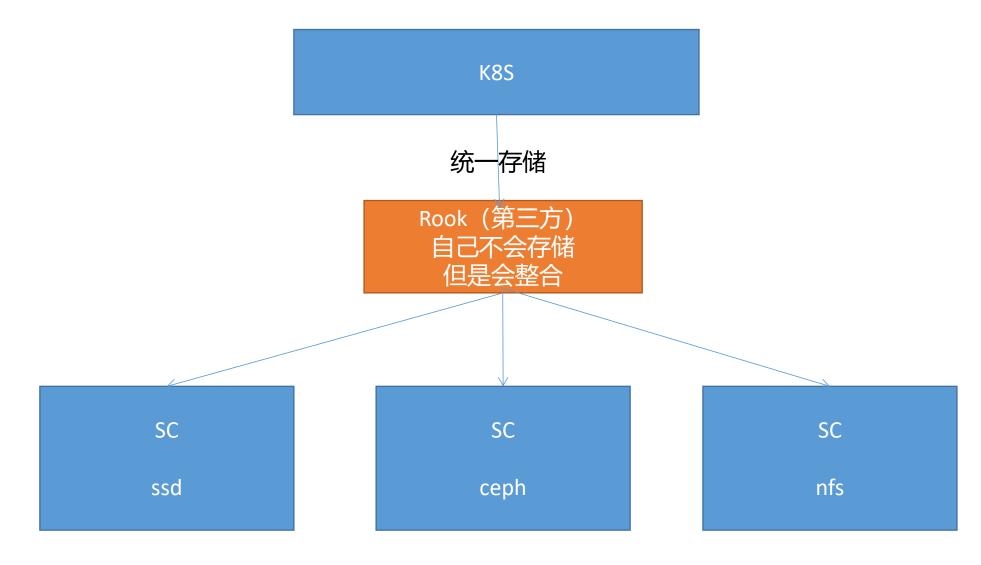
pv

• Pod的开发人员很清楚容器的哪些位置适合挂载I 运维准备好pv池 storageclass进行分组 • 开发人员并不清楚存储技术。存储还要编写详述 my-memory-storage • 要求: Pod文件必须描述每个挂载改怎么挂 2G 2G pv Available 自己绑定的申请书自己用 缺点:资源浪费 不能挂,即使pye删除 运维准备好pv池 运维准备好pv池 pvc 2g storageclass进行分组 storageclass进行分组 my-nfs-storage my-ceph-storage 静态供应 1G 10mb 2G 2G pv pν **7G** pv Available pv

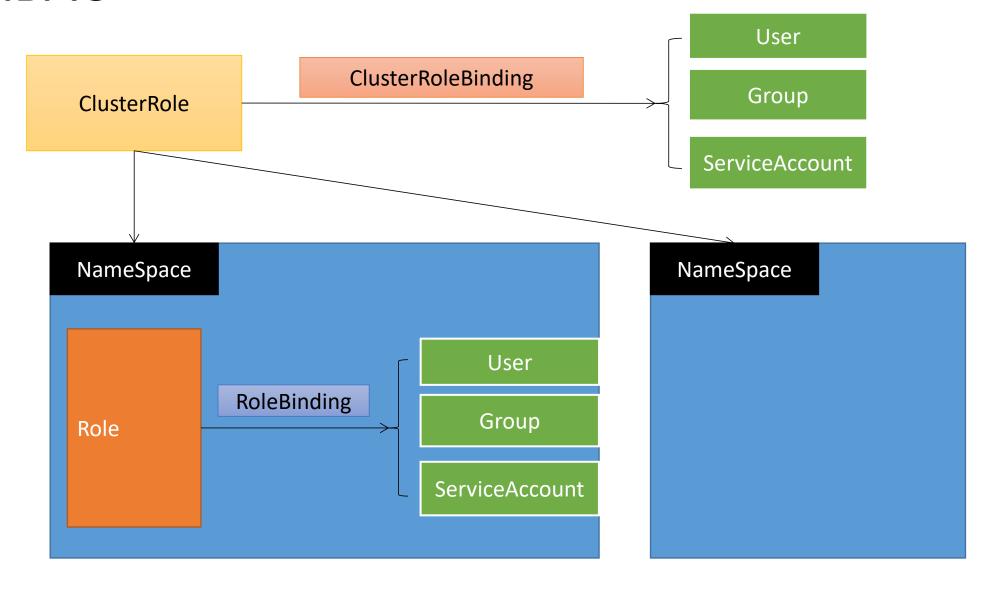
Bound



# 动态供应



#### **RBAC**

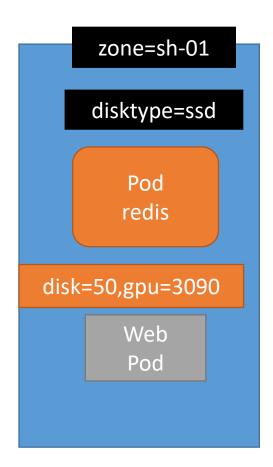


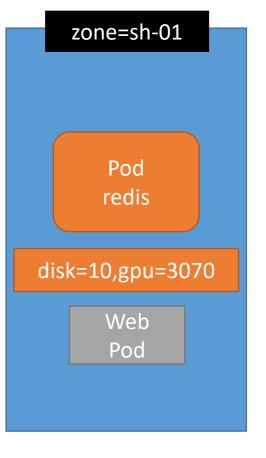
### nodeAffinity

Pod affinity.nodeAffinity.required(必须) affinity.nodeAffinity.preferred(不必须)

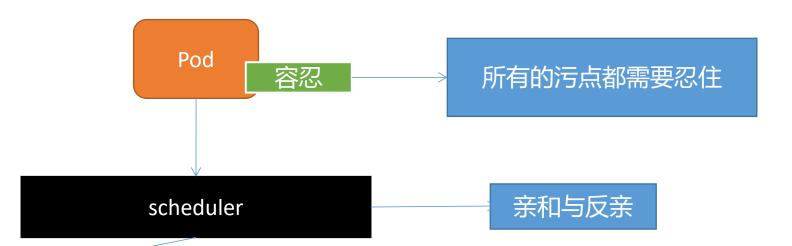
scheduler

zone=01 Pod label Pod label





# 污点与容忍



污点

污点

污点

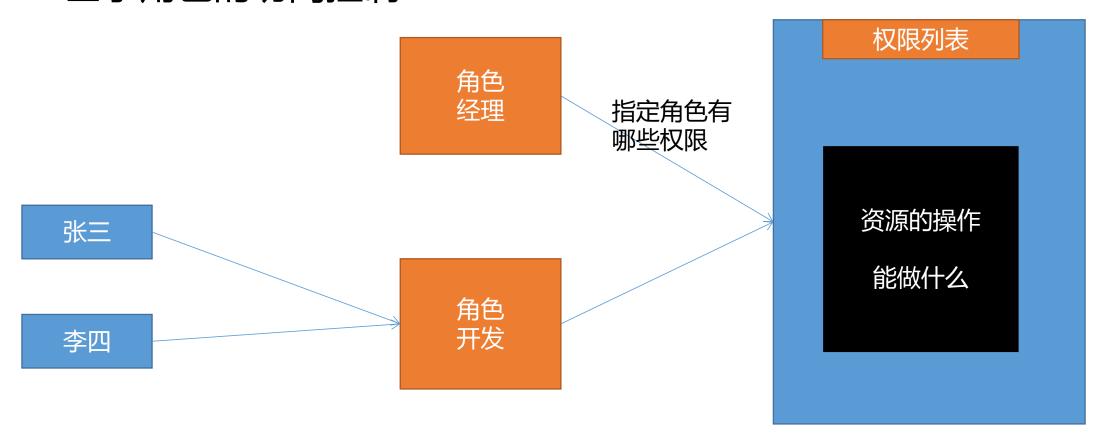
Node

这个节点上 默认不会调 度Pod Node

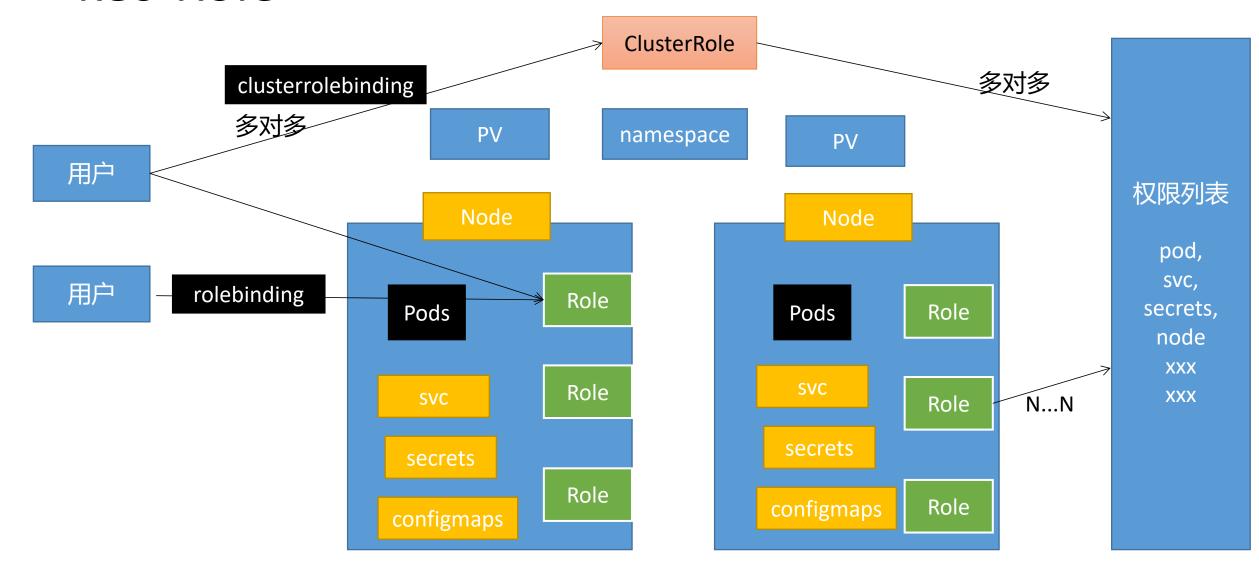
Node

### RBAC (Role-Based-Access-Controller)

• 基于角色的访问控制



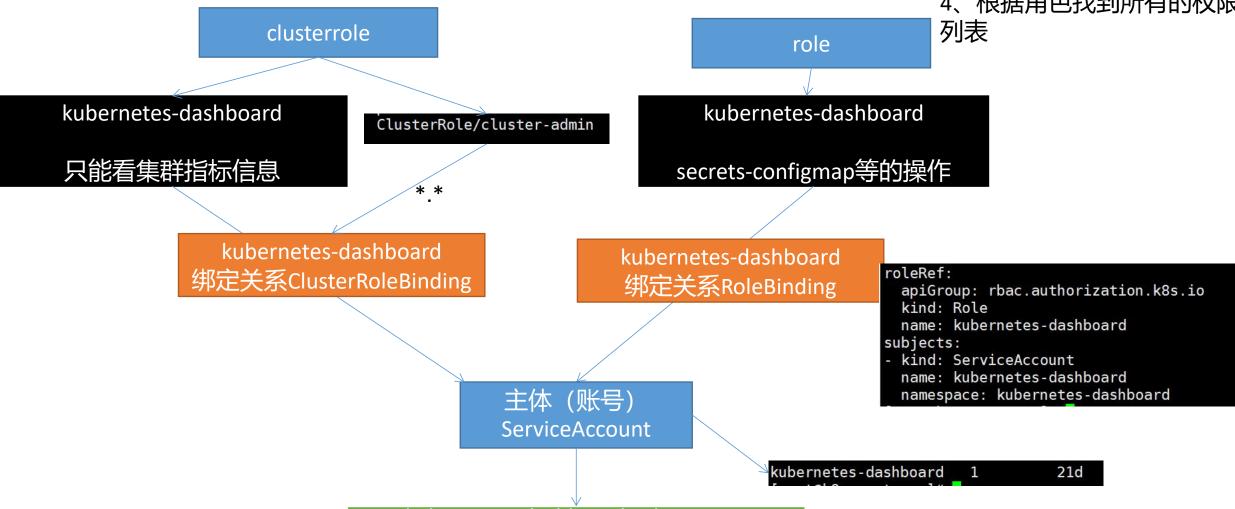
#### k8s-Role



# Dashboard为什么能操作整个集群

- 1、每个账号一个令牌
- 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



使用这个令牌登录即可

kubernetes-dashboard-token-gcxqt

### 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

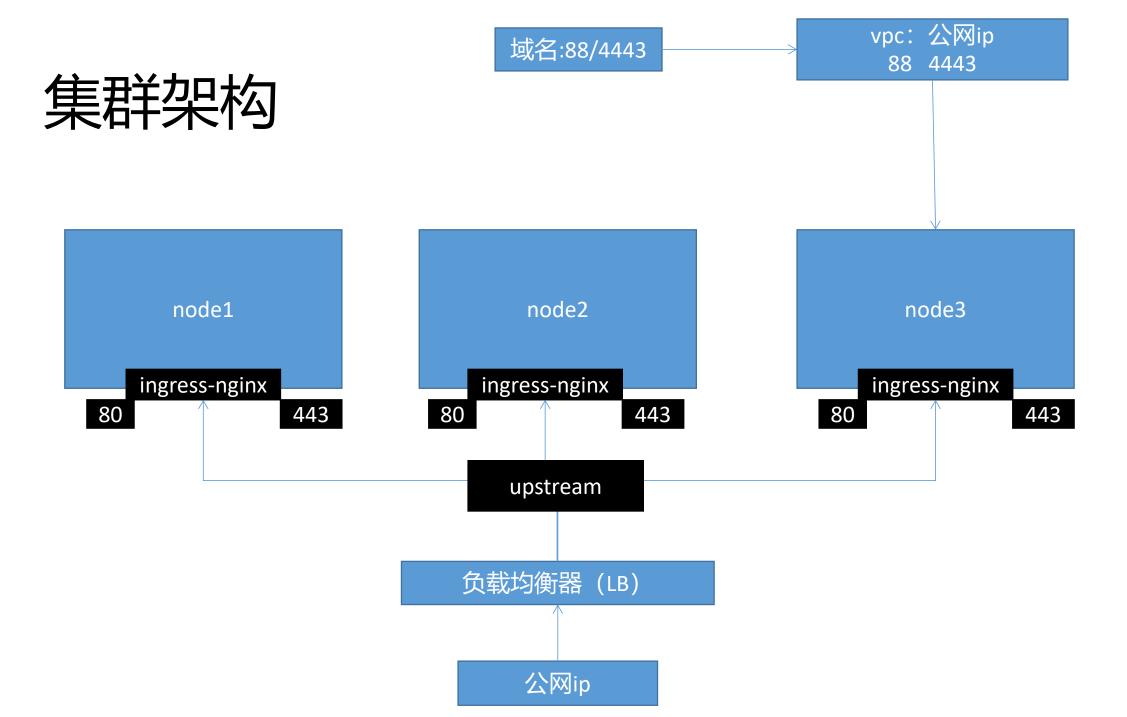
#### Helm

Charts

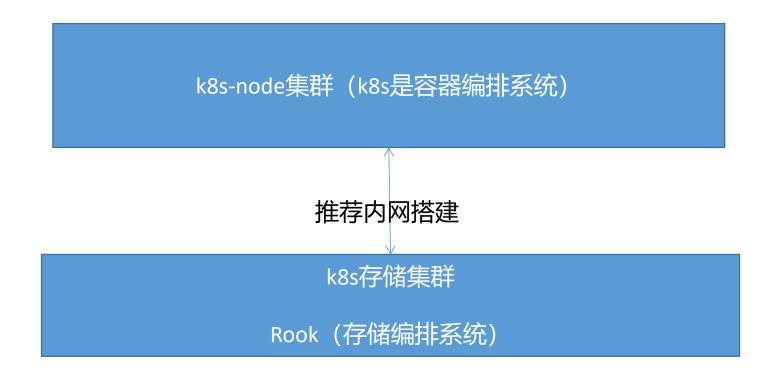
本地Helm 添加Repo 仓库位置2

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

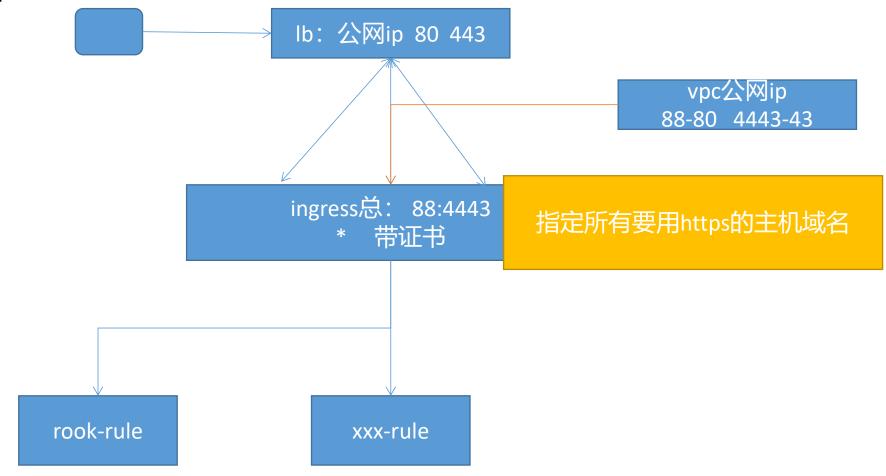
Repository https://artifacthub.io/ bytedance repo atguigu repo alibaba repo



# 存储集群

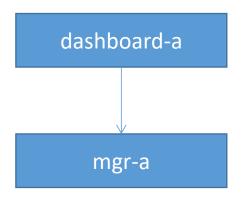


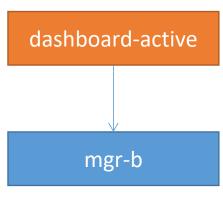
### 网络



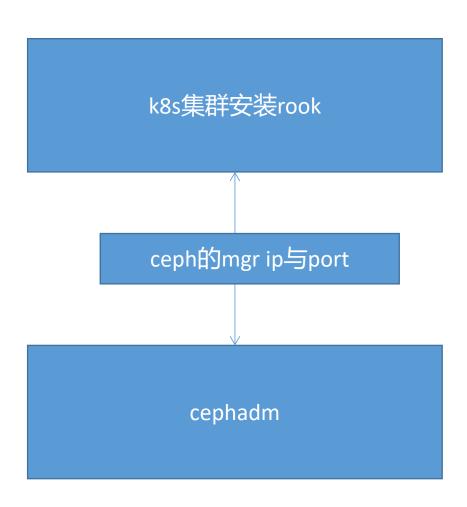
# ceph-mgr的高可用

• a:主 b:备standby

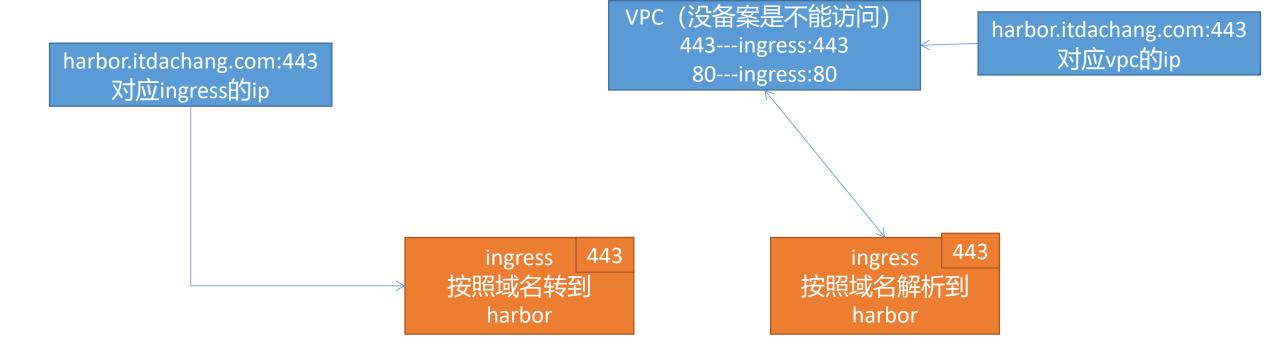




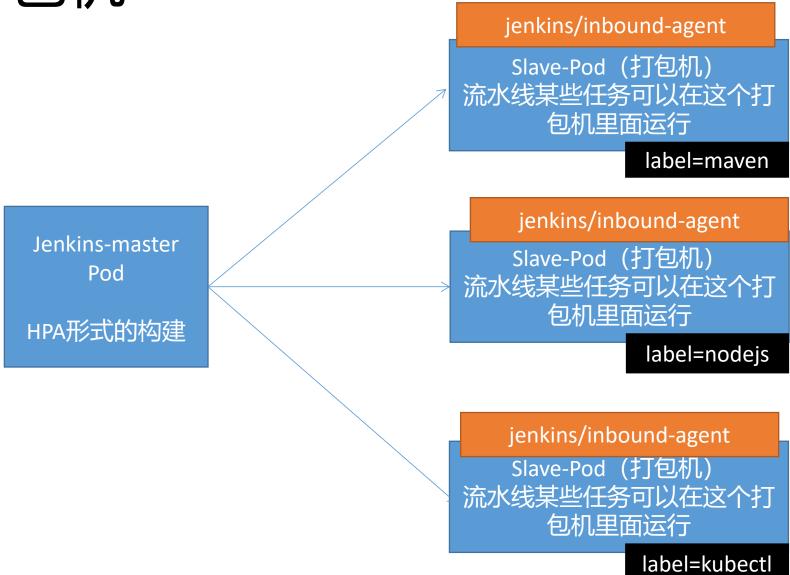
# 集群规划



### ingress

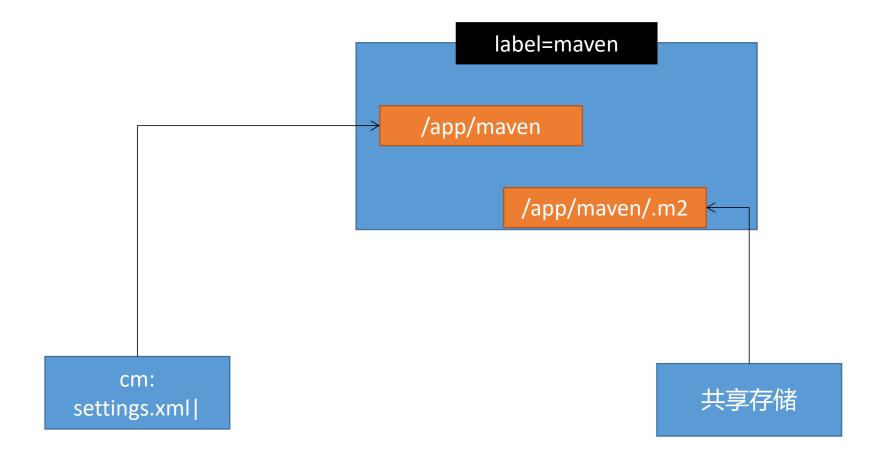


### 打包机



```
多阶段
stages{
stage(){
   agent {
    label: "maven"
```

# 打包机-maven



# 打包机-kubectl

