docker基础

具体的参考，docker篇，这里只是补充

多阶段小镜像

```
#编译环境，生成二进制文件
FROM golang:1.14.2-alpine3.11 as builder
MAINTAINER jett <jettjia@qq.com>
ENV GO111MODULE=on
RUN apk update && \
    apk upgrade && \
    apk add git gcc libc-dev linux-headers
RUN go get -ldflags "-X main.VERSION=$(date -u +%Y%m%d) -s -w"
github.com/xtaci/kcptun/client && go get -ldflags "-X main.VERSION=$(date -u +%Y%m%d) -
s -w" github.com/xtaci/kcptun/server

#只拷贝所需要的二进制文件
FROM alpine:3.11
RUN apk add --no-cache iptables
COPY --from=builder /go/bin /bin
EXPOSE 29900/udp
EXPOSE 12948
```

这个例子，是以golang 1.14 进行打包的，其实主要命令就是go get 会生成文件到GOPATH="/home/chunk/go"，比如以上dockerfile的go get会生成两个文件到GOPATH的bin

```
ls /home/chunk/go/bin/
client  server
```

多阶段构建的目的只要就是缩小docker容器的体积，最终打出来的镜像以FROM alpine:3.11为准，关键在于 COPY --from=builder /go/bin /bin， 只拷贝了/go/bin 整个目录到/bin，即系client和server这个两个二进制文件，从而避免把git gcc libc-dev linux-headers这些包也打入镜像，从而达到最小镜像。

如何运行这个镜像

```
docker run -d -p 1505:1505 -p 29900:29900 xtaci/kcptun client -r xxx.xxx.xxx.xxx:29900
-l :1505 -key test -mtu 1400 -mode fast3
```

https://www.cnblogs.com/csnd/p/12061840.html

https://www.jianshu.com/p/f92000f6cba6

scratch镜像

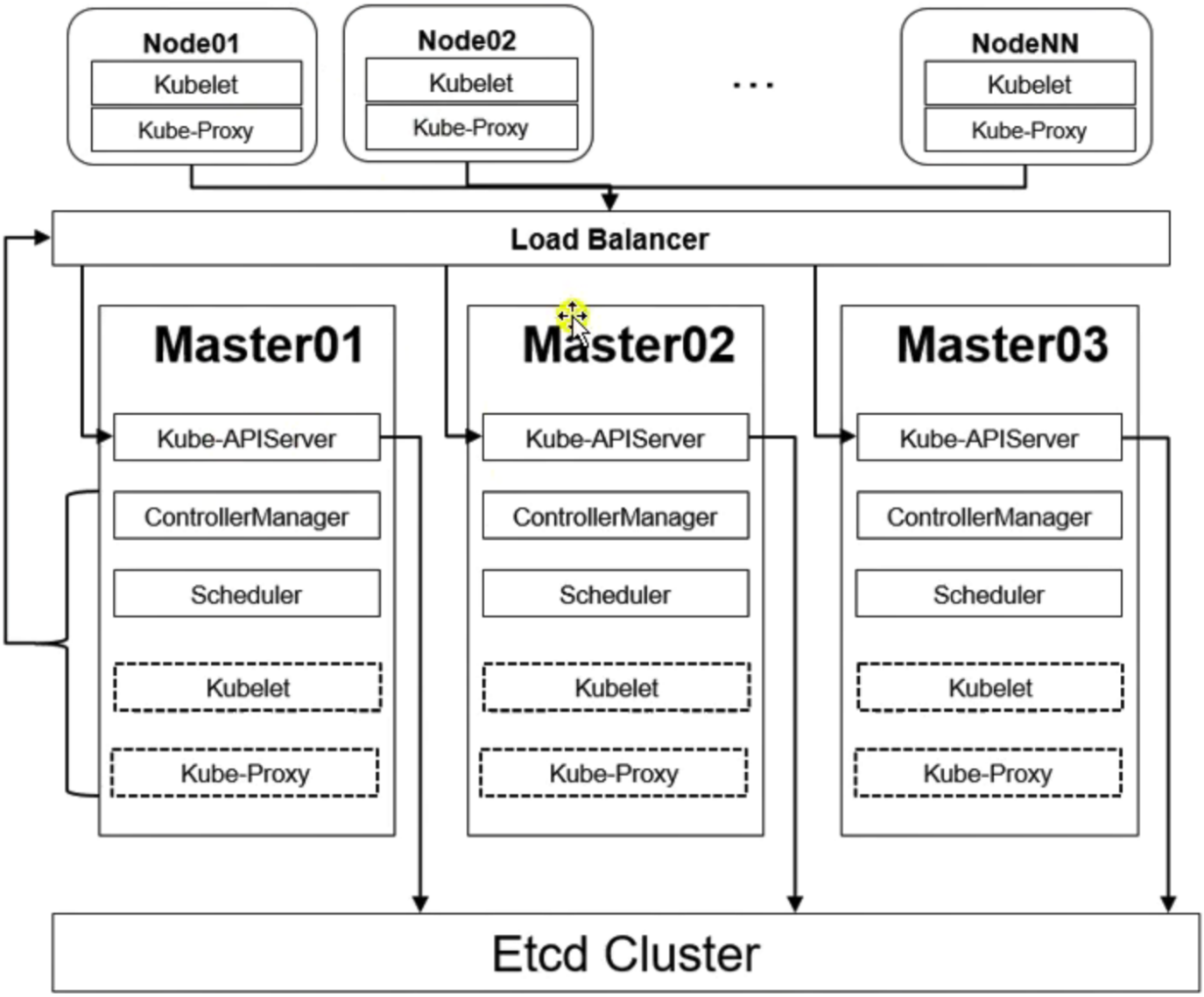
更小的镜像，但是如果有依赖的话，尽量少用。可以用alpine

k8s基础

k8s介绍

k8s是一个全新的基于容器技术的分布式架构解决方案，并且是一个一站式的完备的分布式系统开发和支撑平台。

使用架构



基础组件概述

Master节点组件

Master节点：整个集群的控制中枢.。 组件有：kube-apiserver、kube-controller-manager、kube-scheduler

* kube-apiserver: 集群的控制中枢，各个模块之间信息交互都需要经过 kube-apiserver，同时它也是集群管理、资源配置、整个集群安全机制的入口。

* kube-controller-manager: 集群的状态管理器，保证 Pod 或其他资源到达期望值，也是需要和 apiserver进行通信，在需要的时候创建、更新或删除它所管理的资源。

* kube-scheduler：集群的调度中心，它会根据指定的一些列条件，选择一个或一批最佳的节点，然后部署我们的 Pod。

* etcd：键值数据库，保存一些集群的信息。生产环境部署奇数个节点，3个以上，推荐使用ssd磁盘。

Node节点组件

Worker、Node节点、Minion节点，Node节点组件是指运行在Node节点上，负责具体 Pod 运行时环境的组件。

* kubelet: 负责监听节点上 Pod 的状态，同时负责上报节点和节点上面 Pod 的状态，负责与 Master 节点通信，并管理节点上的 Pod。

* kube-proxy: 负责 Pod 之间的通信和负载均衡，将指定的流量分发到后端正确的机器上。

```
[root@k8s-master01 ~]# netstat -lntp | grep kube-proxy
tcp        0      0 0.0.0.0:31204           0.0.0.0:*               LISTEN
1261/kube-proxy
tcp        0      0 127.0.0.1:10249         0.0.0.0:*               LISTEN
1261/kube-proxy
tcp6       0      0 :::10256                :::*                    LISTEN
1261/kube-proxy
[root@k8s-master01 ~]# curl 127.0.0.1:10249/proxyMode
ipvs
```

ipvs：监听 Master 节点增加和删除 service 以及 endpoint 的消息，调用 Netlink 接口创建相应的 IPVS 规则。通过IPVS规则，将流量转发至相应的Pod上。

iptables: 监听 Master 节点增加和删除 service 以及 endpoint 的消息，对于每一个 Service，他都会创建一个 iptables规则，将 service 的 clusterIP 代理到后端对应的 Pod。

```
# ipvs中，dashboard的演示.
[root@k8s-master01 ~]# ipvsadm -ln
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Flags
  -> RemoteAddress:Port           Forward Weight ActiveConn InActConn
TCP  172.17.0.1:31204 rr
```

```
  -> 172.169.244.196:8443         Masq    1      0          0
TCP  172.169.244.192:31204 rr
  -> 172.169.244.196:8443         Masq    1      0          0
TCP  10.4.7.107:31204 rr
  -> 172.169.244.196:8443         Masq    1      0          0
TCP  10.4.7.236:31204 rr
  -> 172.169.244.196:8443         Masq    1      1          0
TCP  10.96.0.1:443 rr
  -> 10.4.7.107:6443              Masq    1      0          0
  -> 10.4.7.108:6443              Masq    1      0          0
  -> 10.4.7.109:6443              Masq    1      1          0
TCP  10.96.0.10:53 rr
  -> 172.161.125.4:53             Masq    1      0          0
TCP  10.96.0.10:9153 rr
  -> 172.161.125.4:9153           Masq    1      0          0
TCP  10.98.15.94:443 rr
  -> 172.169.244.196:8443         Masq    1      0          0
TCP  10.102.175.175:443 rr
  -> 172.171.14.195:4443          Masq    1      3          0
TCP  10.108.133.81:8000 rr
  -> 172.169.92.67:8000           Masq    1      0          0
TCP  127.0.0.1:31204 rr
  -> 172.169.244.196:8443         Masq    1      0          0
UDP  10.96.0.10:53 rr
  -> 172.161.125.4:53             Masq    1      0          0
[root@k8s-master01 ~]# kubectl get svc -n kubernetes-dashboard
NAME                        TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)
AGE
dashboard-metrics-scraper   ClusterIP   10.108.133.81   <none>        8000/TCP
12h
kubernetes-dashboard        NodePort    10.98.15.94     <none>        443:31204/TCP
12h
[root@k8s-master01 ~]# kubectl get po -n kubernetes-dashboard -owide
NAME                                         READY   STATUS    RESTARTS   AGE   IP
           NODE           NOMINATED NODE   READINESS GATES
dashboard-metrics-scraper-7645f69d8c-6ckjn   1/1     Running   2          12h
172.169.92.67     k8s-master02   <none>           <none>
kubernetes-dashboard-78cb679857-lrjkb        1/1     Running   2          12h
172.169.244.196   k8s-master01   <none>           <none>
```

Calico: 符合 CNI标准的网络插件，给每个 Pod 生成一个唯一的IP地址，并且把每个节点当作一个路由器。

```
[root@k8s-master01 ~]# kubectl get po -n kube-system -owide
NAME                                       READY   STATUS    RESTARTS   AGE   IP
        NODE           NOMINATED NODE   READINESS GATES
calico-kube-controllers-5f6d4b864b-rnw7z   1/1     Running   2          13h
10.4.7.109       k8s-master03   <none>           <none>
calico-node-78fgh                          1/1     Running   2          13h
10.4.7.111       k8s-node02     <none>           <none>
calico-node-bmm4m                          1/1     Running   2          13h
10.4.7.110       k8s-node01     <none>           <none>
calico-node-lz5pw                          1/1     Running   2          13h
10.4.7.109       k8s-master03   <none>           <none>
calico-node-ndhdr                          1/1     Running   2          13h
10.4.7.107       k8s-master01   <none>           <none>
calico-node-pqd7s                          1/1     Running   2          13h
10.4.7.108       k8s-master02   <none>           <none>
coredns-867d46bfc6-h9j7n                   1/1     Running   2          13h
172.161.125.4    k8s-node01     <none>           <none>
metrics-server-595f65d8d5-gzwxf            1/1     Running   2          13h
172.171.14.195   k8s-node02     <none>           <none>
```

CoreDNS: 用于kubernetes集群内部 service的解析，可以让 Pod 把service 名称解析成IP地址，然后通过 service 的IP地址进行链接到对应的应用上。

```
[root@k8s-master01 ~]# kubectl get svc -n kube-system
NAME             TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)
AGE
kube-dns         ClusterIP   10.96.0.10       <none>        53/UDP,53/TCP,9153/TCP
13h
metrics-server   ClusterIP   10.102.175.175   <none>        443/TCP
13h
```

Docker: 容器引擎，负责对容器的管理。

Pod

什么是Pod

pod 是kubernetes 中最小的单元，它由一组、一个或多个容器组成，每个 Pod还包含了一个 Pause容器，Pause 容器是 Pod的父容器，主要负责僵尸进程的回收管理，通过 Pause容器可以使多个容器共享存储、网络、PID、IPC 等。

说明：Pod一般不直接操作，通过 Deployment、StatefulSet、DaemonSet控制。

定义一个 Pod

一份比较完整的yaml文件介绍

```
apiVersion: v1  # 必选，API的版本号
kind: Pod       # 必选，类型Pod
metadata:       # 必选，元数据
  name: nginx   # 必选，符合RFC 1035规范的Pod名称
  namespace: default # 可选，Pod所在的命名空间，不指定默认为default，可以使用-n 指定namespace
  labels:       # 可选，标签选择器，一般用于过滤和区分Pod
    app: nginx
    role: frontend # 可以写多个
  annotations:  # 可选，注释列表，可以写多个
    app: nginx
spec:   # 必选，用于定义容器的详细信息
  initContainers: # 初始化容器，在容器启动之前执行的一些初始化操作
  - command:
    - sh
    - -c
    - echo "I am InitContainer for init some configuration"
    image: busybox
    imagePullPolicy: IfNotPresent
    name: init-container
  containers:   # 必选，容器列表
  - name: nginx # 必选，符合RFC 1035规范的容器名称
    image: nginx:latest          # 必选，容器所用的镜像的地址
    imagePullPolicy: Always      # 可选，镜像拉取策略，IfNotPresent：如果宿主机有这个镜像，那就不需要拉取了. Always：总是拉取，Never：不管是否存储都不拉去
    command: # 可选，容器启动执行的命令
    - nginx
    - -g
    - "daemon off;"
    workingDir: /usr/share/nginx/html       # 可选，容器的工作目录
    volumeMounts:   # 可选，存储卷配置，可以配置多个
    - name: webroot # 存储卷名称
      mountPath: /usr/share/nginx/html # 挂载目录
      readOnly: true         # 只读
    ports:  # 可选，容器需要暴露的端口号列表
    - name: http    # 端口名称
      containerPort: 80     # 端口号
      protocol: TCP # 端口协议，默认TCP
```

```
    env:    # 可选，环境变量配置列表
    - name: TZ      # 变量名
      value: Asia/Shanghai # 变量的值
    - name: LANG
      value: en_US.utf8
    resources:      # 可选，资源限制和资源请求限制
      limits:       # 最大限制设置
        cpu: 1000m
        memory: 1024Mi
      requests:     # 启动所需的资源
        cpu: 100m
        memory: 512Mi
#    startupProbe: # 可选，检测容器内进程是否完成启动。注意三种检查方式同时只能使用一种。
#      httpGet:      # httpGet检测方式，生产环境建议使用httpGet实现接口级健康检查，健康检查由应用
程序提供。
#            path: /api/successStart # 检查路径
#            port: 80
    readinessProbe: # 可选，健康检查。注意三种检查方式同时只能使用一种。
      httpGet:      # httpGet检测方式，生产环境建议使用httpGet实现接口级健康检查，健康检查由应用程
序提供。
            path: / # 检查路径
            port: 80        # 监控端口
    livenessProbe:  # 可选，健康检查
      #exec:        # 执行容器命令检测方式
            #command:
            #- cat
            #- /health
    #httpGet:       # httpGet检测方式
    #   path: /_health # 检查路径
    #   port: 8080
    #   httpHeaders: # 检查的请求头
    #   - name: end-user
    #     value: Jason
      tcpSocket:    # 端口检测方式
            port: 80
      initialDelaySeconds: 60       # 初始化时间
      timeoutSeconds: 2     # 超时时间
      periodSeconds: 5      # 检测间隔
      successThreshold: 1 # 检查成功为2次表示就绪
      failureThreshold: 2 # 检测失败1次表示未就绪
    lifecycle:
      postStart: # 容器创建完成后执行的指令，可以是exec httpGet TCPSocket
        exec:
          command:
          - sh
          - -c
          - 'mkdir /data/ '
      preStop:
        httpGet:
```

```
              path: /
              port: 80
      #  exec:
      #    command:
      #    - sh
      #    - -c
      #    - sleep 9
  restartPolicy: Always   # 可选，默认为Always，容器故障或者没有启动成功，那就自动该容器，
Onfailure: 容器以不为0的状态终止，自动重启该容器，Never:
  #nodeSelector: # 可选，指定Node节点
  #      region: subnet7
  imagePullSecrets:     # 可选，拉取镜像使用的secret，可以配置多个
  - name: default-dockercfg-86258
  hostNetwork: false    # 可选，是否为主机模式，如是，会占用主机端口
  volumes:      # 共享存储卷列表
  - name: webroot # 名称，与上述对应
    emptyDir: {}    # 挂载目录
        #hostPath:              # 挂载本机目录
        #  path: /etc/hosts
```

创建一个容器（maser01上）

```
[root@k8s-master01 ~]# cat > pod.yaml << EOF
apiVersion: v1 # 必选，API的版本号
kind: Pod       # 必选，类型Pod
metadata:       # 必选，元数据
  name: nginx   # 必选，符合RFC 1035规范的Pod名称
  # namespace: default # 可选，Pod所在的命名空间，不指定默认为default，可以使用-n 指定namespace
  labels:       # 可选，标签选择器，一般用于过滤和区分Pod
    app: nginx
    role: frontend # 可以写多个
  annotations:  # 可选，注释列表，可以写多个
    app: nginx
spec:   # 必选，用于定义容器的详细信息
  containers:   # 必选，容器列表
  - name: nginx # 必选，符合RFC 1035规范的容器名称
    image: nginx:1.15.2     # 必选，容器所用的镜像的地址
    imagePullPolicy: IfNotPresent     # 可选，镜像拉取策略，IfNotPresent：如果宿主机有这个镜
像，那就不需要拉取了. Always：总是拉取，Never：不管是否存储都不拉去
    command: # 可选，容器启动执行的命令 ENTRYPOINT, arg --> cmd
    - nginx
    - -g
    - "daemon off;"
    workingDir: /usr/share/nginx/html       # 可选，容器的工作目录
    ports:  # 可选，容器需要暴露的端口号列表
    - name: http    # 端口名称
      containerPort: 80     # 端口号
```

```
      protocol: TCP # 端口协议，默认TCP
    env:    # 可选，环境变量配置列表
    - name: TZ      # 变量名
      value: Asia/Shanghai # 变量的值
    - name: LANG
      value: en_US.utf8
  restartPolicy: Always    # 可选，默认为Always，容器故障或者没有启动成功，那就自动该容器，
Onfailure: 容器以不为0的状态终止，自动重启该容器，Never:
EOF

# 创建一个pod
[root@k8s-master01 ~]# kubectl create -f pod.yaml
pod/nginx created

# 查看刚刚创建的Pod
[root@k8s-master01 ~]# kubectl  get po
NAME      READY   STATUS    RESTARTS   AGE
busybox   1/1     Running   4          17h
nginx     1/1     Running   0          5m24s

[root@k8s-master01 ~]# kubectl  get po --show-labels
NAME      READY   STATUS              RESTARTS   AGE   LABELS
busybox   1/1     Running             4          16h   <none>
nginx     0/1     Running         0          90s   app=nginx,role=frontend

# 删除一个po
[root@k8s-master01 ~]# kubectl delete po nginx
pod "nginx" deleted

# 创建到指定的命名空间
[root@k8s-master01 ~]# kubectl create -f pod.yaml  -n kube-public

# 创建命名空间
kubectl create ns ns_name
```

Pod探针

startupProbe: k8s 1.16版本后新加的探测方式，用于判断容器内应用程序是否启动。如果配置了startupProbe,就会先禁止其他的探测，直到他成功为止，成功后将不会在进行探测。

livenessProbe： 用于探测容器是否运行，入宫探测失败，kubelet 会根据配置的重启策略进行相应的处理。如果没有配置，默认就是success

readinessProbe: 一般用于探测容器内的程序是否健康，它的返回值如果是success，那么代表这个容器已经完成启动，并且程序已经是可以接受流量的状态。

Pod探针的检测方式

ExecAction：在容器内执行一个命令，如果返回值为0，则认为容器健康

TCPSocketAction：通过TCP连接检查容器内的端口是否是通的，如果是通的就认为容器健康

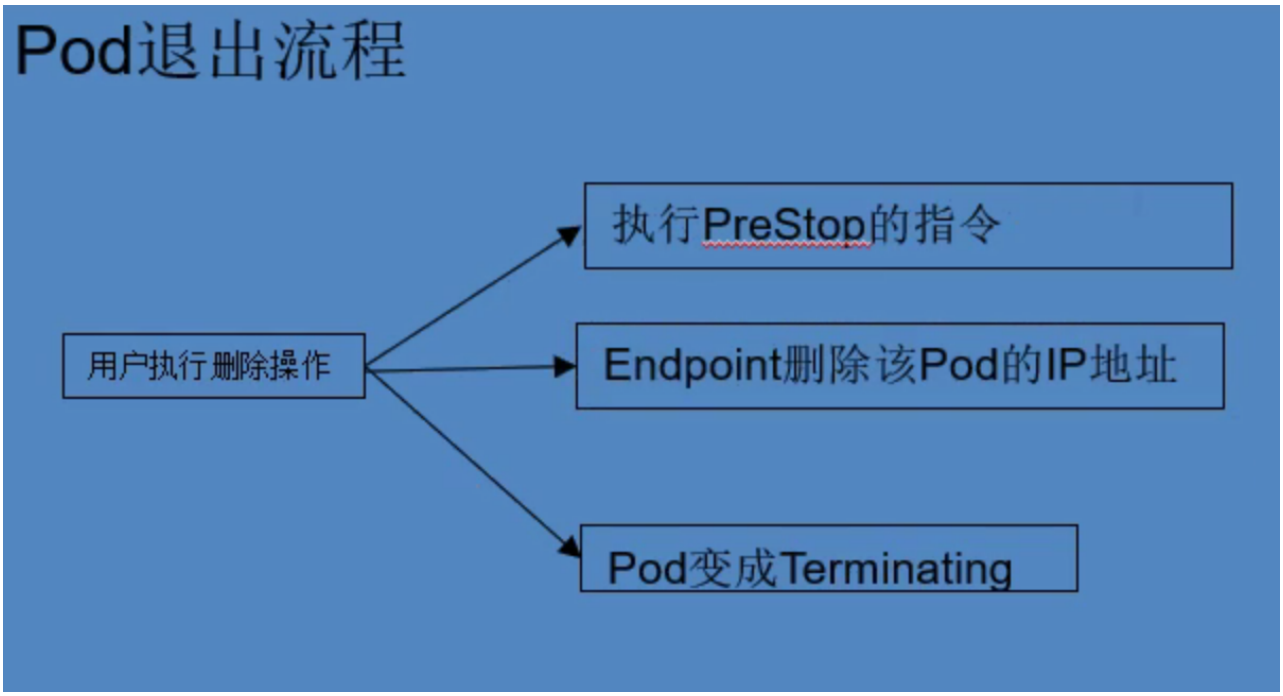
HTTPGetAction：通过应用程序暴露的API地址来检查程序是否是正常的，如果状态码为200~400之间，则认为容器健康

探针检查参数配置

initialDelaySeconds: 60　　# 初始化时间
timeoutSeconds: 2　　# 超时时间
periodSeconds: 5　　# 检测间隔
successThreshold: 1　　# 检查成功为1次表示就绪
failureThreshold: 2　　# 检测失败2次表示未就绪

```
# 查看已有的配置
[root@k8s-master01 ~]# kubectl get deployment -n kube-system
NAME                      READY   UP-TO-DATE   AVAILABLE   AGE
calico-kube-controllers   1/1     1            1           16h
coredns                   1/1     1            1           16h
metrics-server            1/1     1            1           16h
[root@k8s-master01 ~]#
[root@k8s-master01 ~]# kubectl edit deploy coredns -n kube-system
```

Pod退出流程



Prestop：先去请求eureka接口，把自己的IP地址和端口号，进行下线，eureka从注册表中删除该应用的IP地址。然后容器进行sleep 90；kill `pgrep java`

这个时间不一定是90s