- Job
- Cronjob
- Service
- endpoint
- 服务暴露给客户端
  - NodePort
  - Ingress
- 磁盘挂载到容器
  - 卷
- 配置启动参数
  - docker 的命令行参数
  - k8s中覆盖docker的`ENTRYPOINT`和`CMD`
- 环境变量
- ConfigMap
  - config-map-->env-->arg

安装kubernetes集群

kubernetes的安装过程极其复杂,对Linux运维不熟悉的情况下安装kubernetes极为困难,再加上国内无法访问google服务器,我们安装k8s就更加困难

kubeasz项目(https://github.com/easzlab/kubeasz)极大的简化了k8s集群的安装过程,使我们可以离线一键安装k8s集群

准备第一台虚拟机

设置虚拟机cpu

上传离线安装文件

- 将`ansible`目录上传到`/etc/`目录下
- 将`easzup`上传到`/root`目录下

准备离线安装环境

在CentOS7虚拟机中执行下面操作

```
cd ~/

# 下载 kubeasz 的自动化安装脚本文件: easzup,如果已经上传过此文件,则不必执行这一步
export release=2.0.3
curl -C- -fLO --retry 3
https://github.com/easzlab/kubeasz/releases/download/${release}/easzup

# 对easzup文件设置执行权限
chmod +x ./easzup

# 下载离线安装文件,并安装配置docker,
# 如果离线文件已经存在则不会重复下载,
# 离线安装文件存放路径: /etc/ansible
./easzup -D

# 启动kubeasz工具使用的临时容器
./easzup -S

# 进入该容器
docker exec -it kubeasz sh

# 下面命令在容器内执行
# 配置离线安装
cd /etc/ansible
sed -i 's/^INSTALL_SOURCE.*$/INSTALL_SOURCE: "offline"/g' roles/chrony/defaults/main.yml
sed -i 's/^INSTALL_SOURCE.*$/INSTALL_SOURCE: "offline"/g' roles/ex-lb/defaults/main.yml
sed -i 's/^INSTALL_SOURCE.*$/INSTALL_SOURCE: "offline"/g' roles/kube-
node/defaults/main.yml
sed -i 's/^INSTALL_SOURCE.*$/INSTALL_SOURCE: "offline"/g' roles/prepare/defaults/main.yml
exit

# 安装 python,已安装则忽略这一步
yum install python -y
```

导入镜像

为了节省时间,后面课程中使用的docker镜像不用再花时间从网络下载

将课前资料中 images.gz 中的镜像导入 docker

```
docker load -i images.gz
```

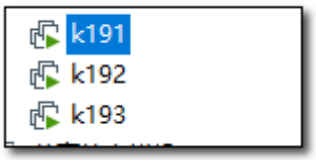
准备三台服务器

准备三台服务器,一台master,两台工作节点,他们的ip地址可以用任意的地址,最好设置为固定ip

下面测试中使用的ip为:

- 192.168.64.191
- 192.168.64.192
- 192.168.64.193

从第一台虚拟机克隆两台虚拟机



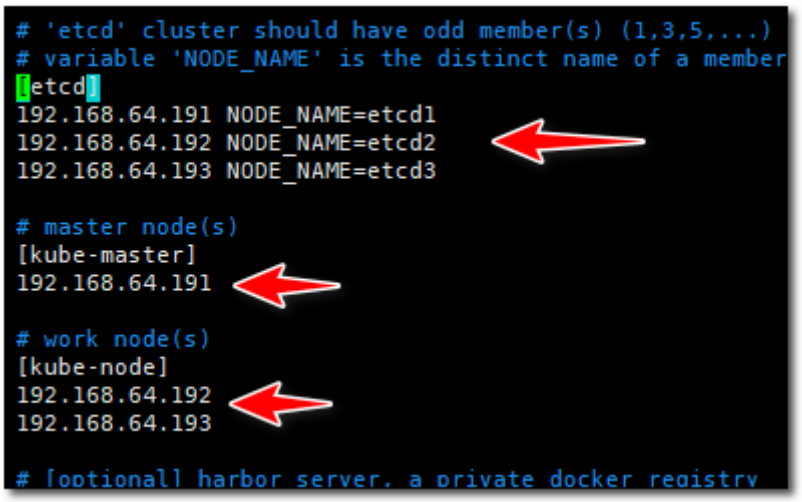
这三台虚拟机,第一台虚拟机作为master,另两台作为工作节点

在master上继续配置安装环境

```
# 安装pip,已安装则忽略这一步
wget -O /etc/yum.repos.d/epel-7.repo https://mirrors.aliyun.com/repo/epel-7.repo
yum install git python-pip -y

# pip安装ansible(国内如果安装太慢可以直接用pip阿里云加速),已安装则忽略这一步
pip install pip --upgrade -i https://mirrors.aliyun.com/pypi/simple/
pip install ansible==2.6.12 netaddr==0.7.19 -i https://mirrors.aliyun.com/pypi/simple/

# 在ansible控制端配置免密码登陆其他节点服务器
ssh-keygen -t ed25519 -N '' -f ~/.ssh/id_ed25519

# 公钥复制到所有节点,包括master自己
# 按提示输入yes和root管理员的密码
ssh-copy-id 192.168.64.191

ssh-copy-id 192.168.64.192

ssh-copy-id 192.168.64.193
```

配置集群服务器的ip

```
cd /etc/ansible && cp example/hosts.multi-node hosts && vim hosts
```




```
# 检查集群主机状态
ansible all -m ping
```

一键安装k8s集群

安装步骤非常多,时间较长,耐心等待安装完成

```
cd /etc/ansible
ansible-playbook 90.setup.yml
```

设置kubectl命令别名

```
# 设置 kubectl 命令别名 k
echo "alias k='kubectl'" >> ~/.bashrc

# 使设置生效
source ~/.bashrc
```

验证安装

```
k get cs
-----------------------------------------------------------
NAME                 STATUS    MESSAGE             ERROR
etcd-1               Healthy   {"health":"true"}
scheduler            Healthy   ok
controller-manager   Healthy   ok
etcd-2               Healthy   {"health":"true"}
etcd-0               Healthy   {"health":"true"}

k get node
```

```
-------------------------------------------------------------------------
NAME             STATUS                     ROLES    AGE     VERSION
192.168.64.191   Ready,SchedulingDisabled   master   5d23h   v1.15.2
192.168.64.192   Ready                      node     5d23h   v1.15.2
192.168.64.193   Ready                      node     5d23h   v1.15.2
```

初步尝试 kubernetes

kubectl run 命令是最简单的部署引用的方式,它自动创建必要组件,这样,我们就先不必深入了解每个组件的结构

使用 ReplicationController 和 pod 部署应用

```
cd ~/

k run \
    --image=luksa/kubia \
    --port=8080 \
    --generator=run/v1 kubia

k get rc
---------------------------------------
NAME    DESIRED   CURRENT   READY   AGE
kubia   1         1         1       24s

k get pods
-----------------------------------------------
NAME          READY   STATUS    RESTARTS   AGE
kubia-9z6kt   1/1     Running   0          28s
```

kubectl run 几个参数的含义

- `--image=luksa/kubia`
  - 镜像名称
- `--port=8080`
  - pod 对外暴露的端口
- `--generator=run/v1 kubia`
  - 创建一个ReplicationController

使用 service 对外暴露 pod

```
k expose \
    rc kubia \
    --type=NodePort \
    --name kubia-http
```

```
k get svc
--------------------------------------------------------------------------------
NAME         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
kubia-http   NodePort    10.68.194.195   <none>        8080:20916/TCP   4s
```

这里创建了一个 service 组件,用来对外暴露pod访问,在所有节点服务器上,暴露了20916端口(随机范围30000-32767),通过此端口,可以访问指定pod的8080端口

访问以下节点服务器的20916端口,都可以访问该应用

注意: 要把端口修改成你生成的随机端口

- http://192.168.64.191:20916/
- http://192.168.64.192:20916/
- http://192.168.64.193:20916/

pod自动伸缩

k8s对应用部署节点的自动伸缩能力非常强,只需要指定需要运行多少个pod,k8s就可以完成pod的自动伸缩

```
# 将pod数量增加到3个
k scale rc kubia --replicas=3

k get po -o wide
--------------------------------------------------------------------------------------------
----------------------
NAME          READY   STATUS    RESTARTS   AGE   IP            NODE             NOMINATED
NODE   READINESS GATES
kubia-q7bg5   1/1     Running   0          10s   172.20.3.29   192.168.64.193   <none>
<none>
kubia-qkcqh   1/1     Running   0          10s   172.20.2.30   192.168.64.192   <none>
<none>
kubia-zlmsn   1/1     Running   0          16m   172.20.3.28   192.168.64.193   <none>
<none>


# 将pod数量减少到1个
k scale rc kubia --replicas=1

# k8s会自动停止两个pod,最终pod列表中会只有一个pod
k get po -o wide
--------------------------------------------------------------------------------------------
----------------------------
NAME          READY   STATUS        RESTARTS   AGE    IP            NODE
NOMINATED NODE   READINESS GATES
kubia-q7bg5   1/1     Terminating   0          6m1s   172.20.3.29   192.168.64.193
<none>           <none>
kubia-qkcqh   1/1     Terminating   0          6m1s   172.20.2.30   192.168.64.192
<none>           <none>
kubia-zlmsn   1/1     Running       0          22m    172.20.3.28   192.168.64.193
<none>           <none>
```

pod

使用部署文件手动部署pod

创建`kubia-manual.yml`部署文件

```
cat <<EOF > kubia-manual.yml
apiVersion: v1               # k8s api版本
kind: Pod                    # 该部署文件用来创建pod资源
metadata:
  name: kubia-manual         # pod名称前缀,后面会追加随机字符串
spec:
  containers:                # 对pod中容器的配置
  - image: luksa/kubia       # 镜像名
    name: kubia              # 容器名
    ports:
    - containerPort: 8080    # 容器暴露的端口
      protocol: TCP
EOF
```

使用部署文件创建pod

```
k create -f kubia-manual.yml

k get po
-----------------------------------------------
NAME           READY   STATUS    RESTARTS   AGE
kubia-manual   1/1     Running   0          19s
```

查看pod的部署文件

```
# 查看pod的部署文件
k get po kubia-manual -o yaml
```

查看pod日志

```
k logs kubia-manual
```

pod端口转发

使用 kubectl port-forward 命令设置端口转发,对外暴露pod.

使用服务器的 8888 端口,映射到 pod 的 8080 端口

```
k port-forward kubia-manual --address localhost,192.168.64.191 8888:8080

# 或在所有网卡上暴露8888端口
k port-forward kubia-manual --address 0.0.0.0 8888:8080
```

在浏览器中访问 http://192.168.64.191:8888/

标签

可以为 pod 指定标签,通过标签可以对 pod 进行分组管理

ReplicationController,ReplicationSet,Service中,都可以通过 Label 来分组管理 pod

创建pod时指定标签

通过`kubia-manual-with-labels.yml`部署文件部署pod

在部署文件中为pod设置了两个自定义标签:`creation_method`和`env`

```
cat <<EOF > kubia-manual-with-labels.yml
apiVersion: v1                  # api版本
kind: Pod                       # 部署的资源类型
metadata:
  name: kubia-manual-v2         # pod名
  labels:                       # 标签设置,键值对形式
    creation_method: manual
    env: prod
spec:
  containers:                   # 容器设置
  - image: luksa/kubia          # 镜像
    name: kubia                 # 容器命名
    ports:                      # 容器暴露的端口
    - containerPort: 8080
      protocol: TCP
EOF
```

使用部署文件创建资源

```
k create -f kubia-manual-with-labels.yml
```

查看pod的标签

列出所有的pod,并显示pod的标签

```
k get po --show-labels
------------------------------------------------------------
NAME              READY   STATUS    RESTARTS   AGE    LABELS
kubia-5rz9h       1/1     Running   0          109s   run=kubia
kubia-manual      1/1     Running   0          52s    <none>
kubia-manual-v2   1/1     Running   0          10s    creation_method=manual,env=prod
```

以列的形式列出pod的标签

```
k get po -L creation_method,env
-------------------------------------------------------------------------------
NAME              READY   STATUS    RESTARTS   AGE     CREATION_METHOD   ENV
kubia-5rz9h       1/1     Running   0          4m19s
kubia-manual      1/1     Running   0          3m22s
kubia-manual-v2   1/1     Running   0          2m40s   manual            prod
```

修改pod的标签

pod `kubia-manual-v2` 的env标签值是`prod`, 我们把这个标签的值修改为 `debug`

修改一个标签的值时,必须指定 `--overwrite` 参数,目的是防止误修改

```
k label po kubia-manual-v2 env=debug --overwrite

k get po -L creation_method,env
-------------------------------------------------------------------------------
NAME              READY   STATUS    RESTARTS   AGE   CREATION_METHOD   ENV
kubia-5rz9h       1/1     Running   0          15m
kubia-manual      1/1     Running   0          14m
kubia-manual-v2   1/1     Running   0          13m   manual            debug
```

为pod `kubia-manual` 设置标签

```
k label po kubia-manual creation_method=manual env=debug
```

为pod `kubia-5rz9h` 设置标签

```
k label po kubia-5rz9h env=debug
```

查看标签设置的结果

```
k get po -L creation_method,env
-------------------------------------------------------------------------
AME              READY   STATUS    RESTARTS   AGE   CREATION_METHOD   ENV
kubia-5rz9h       1/1     Running   0          18m                     debug
kubia-manual      1/1     Running   0          17m   manual            debug
kubia-manual-v2   1/1     Running   0          16m   manual            debug
```

使用标签来查询 pod

查询 `creation_method=manual` 的pod

```
# -l 查询
k get po \
    -l creation_method=manual \
    -L creation_method,env
-------------------------------------------------------------------------
NAME              READY   STATUS    RESTARTS   AGE   CREATION_METHOD   ENV
kubia-manual      1/1     Running   0          28m   manual            debug
kubia-manual-v2   1/1     Running   0          27m   manual            debug
```

查询有 env 标签的 pod

```
# -l 查询
k get po \
    -l env \
    -L creation_method,env
-------------------------------------------------------------------------
NAME              READY   STATUS    RESTARTS   AGE   CREATION_METHOD   ENV
kubia-5rz9h       1/1     Running   0          31m                     debug
kubia-manual      1/1     Running   0          30m   manual            debug
kubia-manual-v2   1/1     Running   0          29m   manual            debug
```

查询 `creation_method=manual` 并且 `env=debug` 的 pod

```
# -l 查询
k get po \
    -l creation_method=manual,env=debug \
    -L creation_method,env
-------------------------------------------------------------------------
NAME              READY   STATUS    RESTARTS   AGE   CREATION_METHOD   ENV
kubia-manual      1/1     Running   0          33m   manual            debug
kubia-manual-v2   1/1     Running   0          32m   manual            debug
```

查询不存在 creation_method 标签的 pod

```
# -l 查询
k get po \
    -l '!creation_method' \
    -L creation_method,env
-------------------------------------------------------------------------
NAME          READY   STATUS    RESTARTS   AGE   CREATION_METHOD   ENV
kubia-5rz9h   1/1     Running   0          36m                     debug
```

其他查询举例:

- `creation_method!=manual`
- `env in (prod,debug)`
- `env notin (prod,debug)`

把pod部署到指定的节点服务器

我们不能直接指定服务器的地址来约束pod部署的节点

通过为node设置标签,在部署pod时,使用节点选择器,来选择把pod部署到匹配的节点服务器

下面为名称为 `192.168.64.193` 的节点服务器,添加标签 `gpu=true`

```
k label node \
    192.168.64.193 \
    gpu=true

k get node \
    -l gpu=true \
    -L gpu
-------------------------------------------------------
NAME             STATUS   ROLES   AGE   VERSION   GPU
192.168.64.193   Ready    node    14d   v1.15.2   true
```

部署文件,其中节点选择器 `nodeSelector` 设置了通过标签 `gpu=true` 来选择节点

```
cat <<EOF > kubia-gpu.yml
apiVersion: v1
kind: Pod
metadata:
  name: kubia-gpu          # pod名
spec:
  nodeSelector:            # 节点选择器,把pod部署到匹配的节点
    gpu: "true"            # 通过标签 gpu=true 来选择匹配的节点
  containers:              # 容器配置
  - image: luksa/kubia     # 镜像
    name: kubia            # 容器名
EOF
```

创建pod `kubia-gpu`,并查看pod的部署节点

```
k create -f kubia-gpu.yml

k get po -o wide
-------------------------------------------------------------------------------------------
----------------------------
NAME              READY   STATUS    RESTARTS   AGE     IP            NODE
NOMINATED NODE   READINESS GATES
kubia-5rz9h       1/1     Running   0          3m13s   172.20.2.35   192.168.64.192
<none>           <none>
kubia-gpu         1/1     Running   0          8m7s    172.20.3.35   192.168.64.193
<none>           <none>
kubia-manual      1/1     Running   0          58m     172.20.3.33   192.168.64.193
<none>           <none>
kubia-manual-v2   1/1     Running   0          57m     172.20.3.34   192.168.64.193
<none>           <none>
```

查看pod `kubia-gpu` 的描述

```
k describe po kubia-gpu
-----------------------------------------------
Name:         kubia-gpu
Namespace:    default
Priority:     0
Node:         192.168.64.193/192.168.64.193
......
```

注解

可以为资源添加注解

注解不能被选择器使用

```
# 注解
k annotate pod kubia-manual tedu.cn/shuoming="foo bar"

k describe po kubia-manual
```

namespace

可以使用命名空间对资源进行组织管理

不同命名空间的资源并不完全隔离,它们之间可以通过网络互相访问

查看命名空间

```
# namespace
k get ns

k get po --namespace kube-system
k get po -n kube-system
```

创建命名空间

新建部署文件`custom-namespace.yml`,创建命名空间,命名为`custom-namespace`

```
cat <<EOF > custom-namespace.yml
apiVersion: v1
kind: Namespace
metadata:
  name: custom-namespace
EOF
```

```
# 创建命名空间
k create -f custom-namespace.yml

k get ns
--------------------------------
NAME               STATUS   AGE
custom-namespace   Active   2s
default            Active   6d
kube-node-lease    Active   6d
kube-public        Active   6d
kube-system        Active   6d
```

将pod部署到指定的命名空间中

创建pod,并将其部署到命名空间`custom-namespace`

```
# 创建 Pod 时指定命名空间
k create \
    -f kubia-manual.yml \
    -n custom-namespace

# 默认访问default命名空间,默认命名空间中不存在pod kubia-manual
k get po kubia-manual

# 访问custom-namespace命名空间中的pod
k get po kubia-manual -n custom-namespace
------------------------------------------------------------
```

```
NAME           READY   STATUS              RESTARTS   AGE
kubia-manual   0/1     ContainerCreating   0          59s
```

删除资源

```
# 按名称删除, 可以指定多个名称
# 例如: k delete po po1 po2 po3
k delete po kubia-gpu

# 按标签删除
k delete po -l creation_method=manual

# 删除命名空间和其中所有的pod
k delete ns custom-namespace

# 删除当前命名空间中所有pod
k delete po --all

# 由于有ReplicationController,所以会自动创建新的pod
[root@master1 ~]# k get po
NAME          READY   STATUS    RESTARTS   AGE
kubia-m6k4d   1/1     Running   0          2m20s
kubia-rkm58   1/1     Running   0          2m15s
kubia-v4cmh   1/1     Running   0          2m15s

# 删除工作空间中所有类型中的所有资源
# 这个操作会删除一个系统Service kubernetes,它被删除后会立即被自动重建
k delete all --all
```

存活探针

有三种存活探针:

- HTTP GET
  返回 2xx 或 3xx 响应码则认为探测成功
- TCP
  与指定端口建立 TCP 连接,连接成功则为成功
- Exec
  在容器内执行任意的指定命令,并检查命令的退出码,退出码为0则为探测成功

HTTP GET 存活探针

luksa/kubia-unhealthy 镜像
在kubia-unhealthy镜像中,应用程序作了这样的设定: 从第6次请求开始会返回500错

在部署文件中,我们添加探针,来探测容器的健康状态.

探针默认每10秒探测一次,连续三次探测失败后重启容器

```
cat <<EOF > kubia-liveness-probe.yml
apiVersion: v1
kind: Pod
metadata:
  name: kubia-liveness               # pod名称
spec:
  containers:
  - image: luksa/kubia-unhealthy     # 镜像
    name: kubia                      # 容器名
    livenessProbe:                   # 存活探针配置
      httpGet:                       # HTTP GET 类型的存活探针
        path: /                      # 探测路径
        port: 8080                   # 探测端口
EOF
```

创建 pod

```
k create -f kubia-liveness-probe.yml

# pod的RESTARTS属性,每过1分半种就会加1
k get po kubia-liveness
-------------------------------------------------
NAME             READY   STATUS    RESTARTS   AGE
kubia-liveness   1/1     Running   0          5m25s
```

查看上一个pod的日志,前5次探测是正确状态,后面3次探测是失败的,则该pod会被删除

```
k logs kubia-liveness --previous
----------------------------------------
Kubia server starting...
Received request from ::ffff:172.20.3.1
Received request from ::ffff:172.20.3.1
Received request from ::ffff:172.20.3.1
Received request from ::ffff:172.20.3.1
Received request from ::ffff:172.20.3.1
Received request from ::ffff:172.20.3.1
Received request from ::ffff:172.20.3.1
Received request from ::ffff:172.20.3.1
```

查看pod描述

```
k describe po kubia-liveness
--------------------------------
......
    Restart Count:  6
    Liveness:       http-get http://:8080/ delay=0s timeout=1s period=10s #success=1
```

```
#failure=3
......
```

- `delay` 0表示容器启动后立即开始探测
- `timeout` 1表示必须在1秒内响应,否则视为探测失败
- `period` 10s表示每10秒探测一次
- `failure` 3表示连续3次失败后重启容器

通过设置 delay 延迟时间,可以避免在容器内应用没有完全启动的情况下就开始探测

```
cat <<EOF > kubia-liveness-probe-initial-delay.yml
apiVersion: v1
kind: Pod
metadata:
  name: kubia-liveness
spec:
  containers:
  - image: luksa/kubia-unhealthy
    name: kubia
    livenessProbe:
      httpGet:
        path: /
        port: 8080
      initialDelaySeconds: 15        # 第一次探测的延迟时间
EOF
```

ReplicationController

RC可以自动化维护多个pod,只需指定pod副本的数量,就可以轻松实现自动扩容缩容

当一个pod宕机,RC可以自动关闭pod,并启动一个新的pod替代它

下面是一个RC的部署文件,设置启动三个kubia容器:

```
cat <<EOF > kubia-rc.yml
apiVersion: v1
kind: ReplicationController        # 资源类型
metadata:
  name: kubia                      # 为RC命名
spec:
  replicas: 3                      # pod副本的数量
  selector:                        # 选择器,用来选择RC管理的pod
    app: kubia                     # 选择标签'app=kubia'的pod,由当前RC进行管理
  template:                        # pod模板,用来创建新的pod
    metadata:
      labels:
        app: kubia                 # 指定pod的标签
    spec:
      containers:                  # 容器配置
```

```
      - name: kubia                # 容器名
        image: luksa/kubia         # 镜像
        ports:
        - containerPort: 8080      # 容器暴露的端口
EOF
```

创建RC

RC创建后,会根据指定的pod数量3,自动创建3个pod

```
k create -f kubia-rc.yml

k get rc
----------------------------------------
NAME    DESIRED   CURRENT   READY   AGE
kubia   3         3         2       2m11s

k get po -o wide
----------------------------------------------------------------------------------------------------
-------------------------------------
NAME             READY   STATUS             RESTARTS   AGE    IP            NODE
NOMINATED NODE   READINESS GATES
kubia-fmtkw      1/1     Running            0          9m2s   172.20.1.7
192.168.64.192   <none>           <none>
kubia-lc5qv      1/1     Running            0          9m3s   172.20.1.8
192.168.64.192   <none>           <none>
kubia-pjs9n      1/1     Running            0          9m2s   172.20.2.11
192.168.64.193   <none>           <none>
```

RC是通过指定的标签`app=kubia`对匹配的pod进行管理的

允许在pod上添加任何其他标签,而不会影响pod与RC的关联关系

```
k label pod kubia-fmtkw type=special

k get po --show-labels
------------------------------------------------------------------------
NAME             READY   STATUS             RESTARTS   AGE     LABELS
kubia-fmtkw      1/1     Running            0          6h31m   app=kubia,type=special
kubia-lc5qv      1/1     Running            0          6h31m   app=kubia
kubia-pjs9n      1/1     Running            0          6h31m   app=kubia
```

但是,如果改变pod的app标签的值,就会使这个pod脱离RC的管理,这样RC会认为这里少了一个pod,那么它会立即创建一个新的pod,来满足我们设置的3个pod的要求

```
k label pod kubia-fmtkw app=foo --overwrite

k get pods -L app
---------------------------------------------------------------------
NAME             READY   STATUS             RESTARTS   AGE     APP
kubia-fmtkw      1/1     Running            0          6h36m   foo
```

```
kubia-lc5qv      1/1     Running            0          6h36m   kubia
kubia-lhj4q      0/1     Pending            0          6s      kubia
kubia-pjs9n      1/1     Running            0          6h36m   kubia
```

修改 pod 模板

pod模板修改后,只影响后续新建的pod,已创建的pod不会被修改

可以删除旧的pod,用新的pod来替代

```
# 编辑 ReplicationController,添加一个新的标签: foo=bar
k edit rc kubia
------------------------------------------------
......
spec:
  replicas: 3
  selector:
    app: kubia
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: kubia
        foo: bar                  # 任意添加一标签
    spec:
......


# 之前pod的标签没有改变
k get pods --show-labels
------------------------------------------------------------------------
NAME             READY   STATUS             RESTARTS   AGE     LABELS
kubia-lc5qv      1/1     Running            0          3d5h    app=kubia
kubia-lhj4q      1/1     Running            0          2d22h   app=kubia
kubia-pjs9n      1/1     Running            0          3d5h    app=kubia


# 通过RC,把pod扩容到6个
# 可以使用前面用过的scale命令来扩容
# k scale rc kubia --replicas=6

# 或者,可以编辑修改RC的replicas属性,修改成6
k edit rc kubia
---------------------
spec:
  replicas: 6        # 从3修改成6,扩容到6个pod
  selector:
    app: kubia


# 新增加的pod有新的标签,而旧的pod没有新标签
k get pods --show-labels
```

```
-------------------------------------------------------------------------
NAME             READY   STATUS    RESTARTS   AGE     LABELS
kubia-8d9jj      0/1     Pending   0          2m23s   app=kubia,foo=bar
kubia-lc5qv      1/1     Running   0          3d5h    app=kubia
kubia-lhj4q      1/1     Running   0          2d22h   app=kubia
kubia-pjs9n      1/1     Running   0          3d5h    app=kubia
kubia-wb8sv      0/1     Pending   0          2m17s   app=kubia,foo=bar
kubia-xp4jv      0/1     Pending   0          2m17s   app=kubia,foo=bar


# 删除 rc, 但不级联删除 pod, 使 pod 处于脱管状态
k delete rc kubia --cascade=false
```

ReplicaSet

ReplicaSet 被设计用来替代 ReplicationController,它提供了更丰富的pod选择功能

以后我们总应该使用 RS, 而不适用 RC, 但在旧系统中仍会使用 RC

```
cat <<EOF > kubia-replicaset.yml
apiVersion: apps/v1              # RS 是 apps/v1中提供的资源类型
kind: ReplicaSet                 # 资源类型
metadata:
  name: kubia                    # RS 命名为 kubia
spec:
  replicas: 3                    # pod 副本数量
  selector:
    matchLabels:                 # 使用 label 选择器
      app: kubia                 # 选取标签是 "app=kubia" 的pod
  template:
    metadata:
      labels:
        app: kubia               # 为创建的pod添加标签 "app=kubia"
    spec:
      containers:
      - name: kubia              # 容器名
        image: luksa/kubia       # 镜像
EOF
```

创建 ReplicaSet

```
k create -f kubia-replicaset.yml

# 之前脱离管理的pod被RS管理
# 设置的pod数量是3,多出的pod会被关闭
k get rs
----------------------------------------
NAME    DESIRED   CURRENT   READY   AGE
kubia   3         3         3       4s
```

```
# 多出的3个pod会被关闭
k get pods --show-labels
-------------------------------------------------------------------------
NAME             READY   STATUS        RESTARTS   AGE     LABELS
kubia-8d9jj      1/1     Pending       0          2m23s   app=kubia,foo=bar
kubia-lc5qv      1/1     Terminating   0          3d5h    app=kubia
kubia-lhj4q      1/1     Terminating   0          2d22h   app=kubia
kubia-pjs9n      1/1     Running       0          3d5h    app=kubia
kubia-wb8sv      1/1     Pending       0          2m17s   app=kubia,foo=bar
kubia-xp4jv      1/1     Terminating   0          2m17s   app=kubia,foo=bar

# 查看RS描述, 与RC几乎相同
k describe rs kubia
```

使用更强大的标签选择器

```
cat <<EOF > kubia-replicaset.yml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: kubia
spec:
  replicas: 4
  selector:
    matchExpressions:       # 表达式匹配选择器
      - key: app            # label 名是 app
        operator: In        # in 运算符
        values:             # label 值列表
          - kubia
          - foo
  template:
    metadata:
      labels:
        app: kubia
    spec:
      containers:
      - name: kubia
        image: luksa/kubia
EOF
```

```
# 先删除现有 RS
k delete rs kubia --cascade=false

# 再创建 RS
k create -f kubia-replicaset.yml
```

可使用的运算符:

- `In`: label与其中一个值匹配
- `NotIn`: label与任何一个值都不匹配
- `Exists`: 包含指定label名称(值任意)

- `DoesNotExists`: 不包含指定的label

清理

```
k delete rs kubia

k get rs
k get po
```

DaemonSet

在每个节点上运行一个 pod,例如资源监控,kube-proxy等

DaemonSet不指定pod数量,它会在每个节点上部署一个pod

```
cat <<EOF > ssd-monitor-daemonset.yml
apiVersion: apps/v1
kind: DaemonSet                         # 资源类型
metadata:
  name: ssd-monitor                     # DS资源命名
spec:
  selector:
    matchLabels:                        # 标签匹配器
      app: ssd-monitor                  # 匹配的标签
  template:
    metadata:
      labels:
        app: ssd-monitor                # 创建pod时,添加标签
    spec:
      containers:                       # 容器配置
        - name: main                    # 容器命名
          image: luksa/ssd-monitor      # 镜像
EOF
```

创建 DS

DS 创建后,会在所有节点上创建pod,包括master

```
k create -f ssd-monitor-daemonset.yml

k get po -o wide
-------------------------------------------------------------------------------------------
--------------------------------
NAME                READY   STATUS    RESTARTS   AGE     IP            NODE
NOMINATED NODE   READINESS GATES
ssd-monitor-g7fjb   1/1     Running   0          57m     172.20.1.12   192.168.64.192
<none>           <none>
ssd-monitor-qk6t5   1/1     Running   0          57m     172.20.2.14   192.168.64.193
<none>           <none>
```

```
ssd-monitor-xxbq8   1/1     Running   0          57m     172.20.0.2    192.168.64.191
<none>           <none>
```

可以在所有选定的节点上部署pod

通过节点的label来选择节点

```
cat <<EOF > ssd-monitor-daemonset.yml
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: ssd-monitor
spec:
  selector:
    matchLabels:
      app: ssd-monitor
  template:
    metadata:
      labels:
        app: ssd-monitor
    spec:
      nodeSelector:                  # 节点选择器
        disk: ssd                    # 选择的节点上具有标签: 'disk=ssd'
      containers:
        - name: main
          image: luksa/ssd-monitor
EOF
```

```
# 先清理
k delete ds ssd-monitor

k create -f ssd-monitor-daemonset.yml
```

查看 DS 和 pod, 看到并没有创建pod,这是因为不存在具有`disk=ssd`标签的节点

```
k get ds

k get po
```

为节点'192.168.64.192'设置标签 `disk=ssd`

这样 DS 会在该节点上立即创建 pod

```
k label node 192.168.64.192 disk=ssd

k get ds
-------------------------------------------------------------------------------------
NAME          DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR   AGE
ssd-monitor   1         1         0       1            0           disk=ssd        37m
```

```
k get po -o wide
-----------------------------------------------------------------------------------------------
----------------------------------------
NAME                READY   STATUS             RESTARTS   AGE     IP            NODE
NOMINATED NODE   READINESS GATES
ssd-monitor-n6d45   1/1     Running            0          16s     172.20.1.13
192.168.64.192   <none>           <none>
```

同样,进一步测试,为节点'192.168.64.193'设置标签 `disk=ssd`

```
k label node 192.168.64.193 disk=ssd
k get ds
k get po -o wide
```

删除'192.168.64.193'节点上的`disk`标签,那么该节点中部署的pod会被立即销毁

```
# 注意删除格式: disk-
k label node 192.168.64.193 disk-
k get ds
k get po -o wide
```

清理

```
k delete ds ssd-monitor
```

Job

Job 用来运行单个任务,任务结束后pod不再重启

```
cat <<EOF > exporter.yml
apiVersion: batch/v1                # Job资源在batch/v1版本中提供
kind: Job                           # 资源类型
metadata:
  name: batch-job                   # 资源命名
spec:
  template:
    metadata:
      labels:
        app: batch-job              # pod容器标签
    spec:
      restartPolicy: OnFailure      # 任务失败时重启
      containers:
        - name: main                # 容器名
          image: luksa/batch-job    # 镜像
EOF
```

创建 job

镜像 batch-job 中的进程,运行120秒后会自动退出

```
k create -f exporter.yml

k get job
----------------------------------------
NAME        COMPLETIONS   DURATION   AGE
batch-job   0/1                      7s

k get po
------------------------------------------------------------
NAME              READY   STATUS              RESTARTS   AGE
batch-job-q97zf   0/1     ContainerCreating   0          7s
```

等待两分钟后,pod中执行的任务退出,再查看job和pod

```
k get job
----------------------------------------
NAME        COMPLETIONS   DURATION   AGE
batch-job   1/1           2m5s       2m16s


k get po
------------------------------------------------------
NAME              READY   STATUS      RESTARTS   AGE
batch-job-q97zf   0/1     Completed   0          2m20s
```

使用Job让pod连续运行5次

先创建第一个pod,等第一个完成后后,再创建第二个pod,以此类推,共顺序完成5个pod

```
cat <<EOF > multi-completion-batch-job.yml
apiVersion: batch/v1
kind: Job
metadata:
  name: multi-completion-batch-job
spec:
  completions: 5                    # 指定完整的数量
  template:
    metadata:
      labels:
        app: batch-job
    spec:
      restartPolicy: OnFailure
      containers:
        - name: main
          image: luksa/batch-job
EOF
```

```
k create -f multi-completion-batch-job.yml
```

共完成5个pod,并每次可以同时启动两个pod

```
cat <<EOF > multi-completion-parallel-batch-job.yml
apiVersion: batch/v1
kind: Job
metadata:
  name: multi-completion-parallel-batch-job
spec:
  completions: 5                    # 共完成5个
  parallelism: 2                    # 可以有两个pod同时执行
  template:
    metadata:
      labels:
        app: batch-job
    spec:
      restartPolicy: OnFailure
      containers:
        - name: main
          image: luksa/batch-job
EOF
```

```
k create -f multi-completion-parallel-batch-job.yml
```

Cronjob

定时和重复执行的任务

cron时间表格式: `"分钟 小时 每月的第几天 月 星期几"`

```
cat <<EOF > cronjob.yml
apiVersion: batch/v1beta1               # api版本
kind: CronJob                           # 资源类型
metadata:
  name: batch-job-every-fifteen-minutes
spec:
  # 0,15,30,45  - 分钟
  # 第一个*   -  每个小时
  # 第二个*   -  每月的每一天
  # 第三个*   -  每月
  # 第四个*   -  每一周中的每一天
  schedule: "0,15,30,45 * * * *"
  jobTemplate:
    spec:
      template:
        metadata:
          labels:
```

```
            app: periodic-batch-job
        spec:
          restartPolicy: OnFailure
          containers:
          - name: main
            image: luksa/batch-job
EOF
```

创建cronjob

```
k create -f cronjob.yml

# 立即查看 cronjob,此时还没有创建pod
k get cj
-------------------------------------------------------------------------------------------
-----
NAME                              SCHEDULE             SUSPEND   ACTIVE   LAST SCHEDULE
AGE
batch-job-every-fifteen-minutes   0,15,30,45 * * * *   False     1        27s
2m17s


# 到0,15,30,45分钟时,会创建一个pod
k get po
-------------------------------------------------------------------------------------------
NAME                                                READY   STATUS      RESTARTS   AGE
batch-job-every-fifteen-minutes-1567649700-vlmdw    1/1     Running     0          36s
```

Service

通过Service资源,为多个pod提供一个单一不变的接入地址

```
cat <<EOF > kubia-svc.yml
apiVersion: v1
kind: Service                # 资源类型
metadata:
  name: kubia                # 资源命名
spec:
  ports:
  - port: 80                 # Service向外暴露的端口
    targetPort: 8080         # 容器的端口
  selector:
    app: kubia               # 通过标签,选择名为kubia的所有pod
EOF
```

```
k create -f kubia-svc.yml

k get svc
-------------------------------------------------------------------------
```

```
NAME         TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)   AGE
kubernetes   ClusterIP   10.68.0.1      <none>        443/TCP   2d11h
kubia        ClusterIP   10.68.163.98   <none>        80/TCP    5s
```

从内部网络访问Service

用`kubtctl exec`命令进入一个容器,执行`curl -s http://10.68.163.98`来访问Service

执行多次会看到,Service会在多个pod中轮训发送请求

```
k exec kubia-5zm2q -- curl -s http://10.68.163.98

# [root@localhost ~]# k exec kubia-5zm2q -- curl -s http://10.68.163.98
# You've hit kubia-xdj86
# [root@localhost ~]# k exec kubia-5zm2q -- curl -s http://10.68.163.98
# You've hit kubia-xmtq2
# [root@localhost ~]# k exec kubia-5zm2q -- curl -s http://10.68.163.98
# You've hit kubia-5zm2q
# [root@localhost ~]# k exec kubia-5zm2q -- curl -s http://10.68.163.98
# You've hit kubia-xdj86
# [root@localhost ~]# k exec kubia-5zm2q -- curl -s http://10.68.163.98
# You've hit kubia-xmtq2
```

回话亲和性

来自同一个客户端的请求,总是发给同一个pod

```
cat <<EOF > kubia-svc-clientip.yml
apiVersion: v1
kind: Service
metadata:
  name: kubia-clientip
spec:
  sessionAffinity: ClientIP        # 回话亲和性使用ClientIP
  ports:
  - port: 80
    targetPort: 8080
  selector:
    app: kubia
EOF
```

```
k create -f kubia-svc-clientip.yml

k get svc
-------------------------------------------------------------------------
NAME             TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)   AGE
kubernetes       ClusterIP   10.68.0.1      <none>        443/TCP   2d12h
kubia            ClusterIP   10.68.163.98   <none>        80/TCP    38m
kubia-clientip   ClusterIP   10.68.72.120   <none>        80/TCP    2m15s

# 进入kubia-5zm2q容器,向Service发送请求
```

```
# 执行多次会看到,每次请求的都是同一个pod
k exec kubia-5zm2q -- curl -s http://10.68.72.120
```

在pod中,可以通过一个环境变量来获知Service的ip地址

该环境变量在旧的pod中是不存在的,我们需要先删除旧的pod,用新的pod来替代

```
k delete po --all

k get po
-----------------------------------------------
NAME          READY   STATUS    RESTARTS   AGE
kubia-k66lz   1/1     Running   0          64s
kubia-vfcqv   1/1     Running   0          63s
kubia-z257h   1/1     Running   0          63s
```

```
k exec kubia-k66lz env
---------------------------------------------------------------------
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
HOSTNAME=kubia-k66lz
KUBIA_SERVICE_PORT=80                           # kubia服务的端口
KUBIA_PORT=tcp://10.68.163.98:80
KUBIA_CLIENTIP_SERVICE_PORT=80                  # kubia-clientip服务的端口
KUBIA_CLIENTIP_PORT_80_TCP=tcp://10.68.72.120:80
KUBIA_CLIENTIP_PORT_80_TCP_PROTO=tcp
KUBERNETES_SERVICE_HOST=10.68.0.1
KUBERNETES_PORT_443_TCP=tcp://10.68.0.1:443
KUBIA_SERVICE_HOST=10.68.163.98                 # kubia服务的ip
KUBIA_CLIENTIP_SERVICE_HOST=10.68.72.120        # kubia-clientip服务的ip
......
```

通过`全限定域名`来访问Service

```
# 进入一个容器
k exec -it kubia-k66lz bash

ping kubia
curl http://kubia
curl http://kubia.default
curl http://kubia.default.svc.cluster.local
```

endpoint

endpoint是在Service和pod之间的一种资源

一个endpoint资源,包含一组pod的地址列表

```
# 查看kubia服务的endpoint
k describe svc kubia
------------------------------
......
Endpoints:         172.20.2.40:8080,172.20.3.57:8080,172.20.3.58:8080
......

# 查看所有endpoint
k get ep
------------------------------------------------------------------------
NAME             ENDPOINTS                                            AGE
kubia            172.20.2.40:8080,172.20.3.57:8080,172.20.3.58:8080   95m
kubia-clientip   172.20.2.40:8080,172.20.3.57:8080,172.20.3.58:8080   59m

# 查看名为kubia的endpoint
k get ep kubia
```

不含pod选择器的服务,不会创建 endpoint

```
cat <<EOF > external-service.yml
apiVersion: v1
kind: Service
metadata:
  name: external-service        #  Service命名
spec:
  ports:
  - port: 80
EOF
```

创建endpoint关联到Service,它的名字必须与Service同名

```
cat <<EOF > external-service-endpoints.yml
apiVersion: v1
kind: Endpoints                 # 资源类型
metadata:
  name: external-service        # 名称要与Service名相匹配
subsets:
- addresses:                    # 包含的地址列表
  - ip: 120.52.99.224           # 中国联通的ip地址
  - ip: 117.136.190.162         # 中国移动的ip地址
  ports:
  - port: 80                    # 目标服务的的端口
EOF
```

```
# 进入一个容器
k exec -it kubia-k66lz bash

# 访问 external-service
# 多次访问,会在endpoints地址列表中轮训请求
curl http://external-service
```

通过`完全限定域名`访问外部服务

```
cat <<EOF > external-service-externalname.yml
apiVersion: v1
kind: Service
metadata:
  name: external-service-externalname
spec:
  type: ExternalName
  externalName: www.chinaunicom.com.cn      # 域名
  ports:
  - port: 80
EOF
```

创建服务

```
k create -f external-service-externalname.yml

# 进入一个容器
k exec -it kubia-k66lz bash

# 访问 external-service-externalname
curl http://external-service-externalname
```

服务暴露给客户端

前面创建的Service只能在集群内部网络中访问,那么怎么让客户端来访问Service呢?

三种方式

- NodePort
  - 每个节点都开放一个端口
- LoadBalance
  - NodePort的一种扩展,负载均衡器需要云基础设施来提供
- Ingress

NodePort

在每个节点(包括master),都开放一个相同的端口,可以通过任意节点的端口来访问Service

端口的默认范围是 30000-32767

```
cat <<EOF > kubia-svc-nodeport.yml
apiVersion: v1
kind: Service
```

```
metadata:
  name: kubia-nodeport
spec:
  type: NodePort           # 在每个节点上开放访问端口
  ports:
  - port: 80               # 集群内部访问该服务的端口
    targetPort: 8080       # 容器的端口
    nodePort: 30123        # 外部访问端口
  selector:
    app: kubia
EOF
```

创建并查看 Service

```
k create -f kubia-svc-nodeport.yml

k get svc kubia-nodeport
--------------------------------------------------------------------------------
NAME             TYPE       CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
kubia-nodeport   NodePort   10.68.140.119   <none>        80:30123/TCP   14m
```

可以通过任意节点的`30123`端口来访问 Service

- http://192.168.64.191:30123
- http://192.168.64.192:30123
- http://192.168.64.193:30123

Ingress

一个 Ingress 资源就可以同时向外暴露多个 Service

- 可以通过子路径暴露多个Service

```
  - host: kubia.example.com
    http:
      paths:
      - path: /kubia           # 子路径
        backend:
          serviceName: kubia   # 暴露的Service
          servicePort: 80
      - path: /foo             # 子路径
        backend:
          serviceName: bar     # 暴露的Service
          servocePort: 80
```

- 通过不同域名来暴露多个Service

```
spec:
  rules:
  - host: foo.example.com    # 用域名暴露Service
    http:
      paths:
      - path: /
        backend:
          serviceName: foo
          servicePort: 80
  - host: bar.example.com    # 用域名暴露Service
    http:
      paths:
      - path: /
        backend:
          serviceName: bar
          servicePort: 80
```

用 Ingress 暴露 kubia-nodeport, 通过域名 `kubia.example.com` 访问时, 会把请求转发到端口 80 上的kubia-nodeport服务

需要修改 hosts 文件,添加域名映射

```
192.168.64.192 kubia.example.com
```

```
cat <<EOF > kubia-ingress.yml
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: kubia
spec:
  rules:
  - host: kubia.example.com
    http:
      paths:
      - path: /
        backend:
          serviceName: kubia-nodeport
          servicePort: 80
EOF
```

```
k create -f kubia-ingress.yml

# 查看 ingress
k get ing
--------------------------------------------------
NAME    HOSTS               ADDRESS   PORTS   AGE
kubia   kubia.example.com             80      15m

# 创建后,用浏览器访问  http://kubia.example.com/
```

通过 Ingress 转发 HTTPS 访问

```
# 创建私钥
openssl genrsa -out tls.key 2048

# 创建证书
openssl req -new -x509 -key tls.key -out tls.cert -days 360 -subj /CN=kubia.example.com

# 创建 Secret
k create secret tls tls-secret --cert=tls.cert --key=tls.key
```

```
cat <<EOF > kubia-ingress-tls.yml
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: kubia
spec:
  tls:                                # tls安全配置
  - hosts:
    - kubia.example.com
    secretName: tls-secret            # 从这个 secret 获得私钥和证书
  rules:
  - host: kubia.example.com
    http:
      paths:
      - path: /
        backend:
          serviceName: kubia-nodeport
          servicePort: 80
EOF
```

```
# 删除前面创建的 ingress
k delete ing kubia

# 重新创建新的 ingress
k create -f kubia-ingress-tls.yml

# 用浏览器访问下面链接:
# http://kubia.example.com/
# https://kubia.example.com/

# 注意: https访问时,浏览器会有安全性警告,可以在高级中选择继续访问
```

磁盘挂载到容器

卷

卷的类型:

- emptyDir: 简单的空目录
- hostPath: 工作节点中的磁盘路径
- gitRepo: 从git克隆的本地仓库
- nfs: nfs共享文件系统

创建包含两个容器的pod, 它们共享同一个卷

```
cat <<EOF > fortune-pod.yml
apiVersion: v1
kind: Pod
metadata:
  name: fortune
  labels:
    app: fortune
spec:
  containers:
  - image: luksa/fortune                # 镜像名
    name: html-genrator                 # 容器名
    volumeMounts:
    - name: html                        # 卷名为 html
      mountPath: /var/htdocs            # 容器中的挂载路径
  - image: nginx:alpine                 # 第二个镜像名
    name: web-server                    # 第二个容器名
    volumeMounts:
    - name: html                        # 相同的卷 html
      mountPath: /usr/share/nginx/html  # 在第二个容器中的挂载路径
      readOnly: true                    # 设置为只读
    ports:
    - containerPort: 80
      protocol: TCP
  volumes:                              # 卷
  - name: html                          # 为卷命名
    emptyDir: {}                        # emptyDir类型的卷
EOF
```

```
k create -f fortune-pod.yml

k get po
```

创建Service, 通过这个Service访问pod的80端口

```
cat <<EOF > fortune-svc.yml
apiVersion: v1
kind: Service
metadata:
  name: fortune
spec:
  type: NodePort
```

```
  ports:
  - port: 8088
    targetPort: 80
    nodePort: 38088
  selector:
    app: fortune
EOF
```

```
k create -f fortune-svc.yml

k get svc

# 用浏览器访问  http://192.168.64.191:38088/
```

NFS 文件系统

在 master 节点 192.168.64.191 上创建 nfs 目录 `/etc/nfs_data`, 并允许 1921.68.64 网段的主机共享访问这个目录

```
# no_root_squash: 服务器端使用root权限

cat <<EOF > /etc/exports
/etc/nfs_data    192.168.64.0/24(rw,async,no_root_squash)
EOF
```

```
systemctl enable nfs
systemctl enable rpcbind
systemctl start nfs
systemctl start rpcbind
```

尝试在客户端主机上,例如192.168.64.192,挂载远程的nfs目录

```
# 在客户端，挂载服务器的 nfs 目录
mount -t nfs 192.168.64.191:/etc/nfs_data /etc/web_dir/
```

持久化存储

创建 PersistentVolume - 持久卷资源

```
cat <<EOF > mongodb-pv.yml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: mongodb-pv
spec:
  capacity:
    storage: 1Gi                          # 定义持久卷大小
```

```
  accessModes:
    - ReadWriteOnce                     # 只允许被一个客户端挂载为读写模式
    - ReadOnlyMany                      # 可以被多个客户端挂载为只读模式
  persistentVolumeReclaimPolicy: Retain   # 当声明被释放,持久卷将被保留
  nfs:                                  # nfs远程目录定义
    path: /etc/nfs_data
    server: 192.168.64.191
EOF
```

```
k get pv
-------------------------------------------------------------------------------------------
-----------------
NAME         CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS      CLAIM   STORAGECLASS
REASON   AGE
mongodb-pv   1Gi        RWO,ROX        Retain           Available
4s
```

持久卷声明

使用持久卷声明,使应用与底层存储技术解耦

```
cat <<EOF > mongodb-pvc.yml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mongodb-pvc
spec:
  resources:
    requests:
      storage: 1Gi           # 申请1GiB存储空间
  accessModes:
    - ReadWriteOnce          # 允许单个客户端读写
  storageClassName: ""       # 参考动态配置章节
EOF
```

```
k get pvc
-------------------------------------------------------------------------------
NAME          STATUS   VOLUME       CAPACITY   ACCESS MODES   STORAGECLASS   AGE
mongodb-pvc   Bound    mongodb-pv   1Gi        RWO,ROX                       3s
```

```
cat <<EOF > mongodb-pod-pvc.yml
apiVersion: v1
kind: Pod
metadata:
  name: mongodb
spec:
  containers:
  - image: mongo
```

```
    name: mongodb
    securityContext:
      runAsUser: 0
    volumeMounts:
    - name: mongodb-data
      mountPath: /data/db
    ports:
    - containerPort: 27017
      protocol: TCP
  volumes:
  - name: mongodb-data
    persistentVolumeClaim:
      claimName: mongodb-pvc      # 引用之前创建的"持久卷声明"
EOF
```

验证 pod 中加挂载了 nfs 远程目录作为持久卷

```
k exec -it mongodb mongo

use mystore
db.foo.insert({name:'foo'})
db.foo.find()
```

查看在 nfs 远程目录中的文件

```
cd /etc/nfs_data
ls
```

配置启动参数

docker 的命令行参数

Dockerfile中定义命令和参数的指令

- `ENTRYPOINT` 启动容器时,在容器内执行的命令
- `CMD` 对启动命令传递的参数

`CMD` 可以在 `docker run` 命令中进行覆盖

例如:

```
......
ENTRYPOINT ["java", "-jar", "/opt/sp05-eureka-0.0.1-SNAPSHOT.jar"]
CMD ["--spring.profiles.active=eureka1"]
```

启动容器时,可以执行:

```
docker run <image>
```

或者启动容器时覆盖CMD

```
docker run <image> --spring.profiles.active=eureka2
```

k8s中覆盖docker的 `ENTRYPOINT` 和 `CMD`

- `command` 可以覆盖 `ENTRYPOINT`
- `args` 可以覆盖 `CMD`

在镜像 `luksa/fortune:args` 中,设置了自动生成内容的间隔时间参数为10秒

```
......
CMD ["10"]
```

可以通过k8s的 `args` 来覆盖docker的 `CMD`

```
cat <<EOF > fortune-pod-args.yml
apiVersion: v1
kind: Pod
metadata:
  name: fortune
  labels:
    app: fortune
spec:
  containers:

  - image: luksa/fortune:args
    args: ["2"]                  # docker镜像中配置的CMD是10,这里用args把这个值覆盖成2
    name: html-genrator
    volumeMounts:
    - name: html
      mountPath: /var/htdocs

  - image: nginx:alpine
    name: web-server
    volumeMounts:
    - name: html
      mountPath: /usr/share/nginx/html
      readOnly: true
    ports:
    - containerPort: 80
      protocol: TCP

  volumes:
```

```
  - name: html
    emptyDir: {}
EOF
```

```
k create -f fortune-pod-args.yml

# 查看pod
k get po -o wide
-----------------------------------------------------------------------------------------------
---------------------
NAME      READY   STATUS    RESTARTS   AGE   IP            NODE              NOMINATED
NODE   READINESS GATES
fortune   2/2     Running   0          34s   172.20.2.55   192.168.64.192   <none>
<none>
```

重复地执行curl命令,访问该pod,会看到数据每2秒刷新一次

注意要修改成你的pod的ip

```
curl http://172.20.2.55
```

环境变量

在镜像`luksa/fortune:env`中通过环境变量`INTERVAL`来指定内容生成的间隔时间

下面配置中,通过`env`配置,在容器中设置了环境变量`INTERVAL`的值

```
cat <<EOF > fortune-pod-env.yml
apiVersion: v1
kind: Pod
metadata:
  name: fortune
  labels:
    app: fortune
spec:
  containers:

  - image: luksa/fortune:env
    env:                          # 设置环境变量 INTERVAL=5
    - name: INTERVAL
      value: "5"
    name: html-genrator
    volumeMounts:
    - name: html
      mountPath: /var/htdocs

  - image: nginx:alpine
    name: web-server
    volumeMounts:
```

```
    - name: html
      mountPath: /usr/share/nginx/html
      readOnly: true
    ports:
    - containerPort: 80
      protocol: TCP

  volumes:
  - name: html
    emptyDir: {}
EOF
```

```
k delete po fortune
k create -f fortune-pod-env.yml

# 查看pod
k get po -o wide
-------------------------------------------------------------------------------------------
--------------------
NAME      READY   STATUS    RESTARTS   AGE   IP            NODE              NOMINATED
NODE   READINESS GATES
fortune   2/2     Running   0          8s    172.20.2.56   192.168.64.192   <none>
<none>

# 进入pod
k exec -it fortune bash
# 查看pod的环境变量
env
------------
INTERVAL=5
......

# 从pod推出,回到宿主机
exit
```

重复地执行curl命令,访问该pod,会看到数据每5秒刷新一次

注意要修改成你的pod的ip

```
curl http://172.20.2.56
```

ConfigMap

通过ConfigMap资源,可以从pod中把环境变量配置分离出来,是环境变量配置与pod解耦

可以从命令行创建ConfigMap资源:

```
# 直接命令行创建
k create configmap fortune-config --from-literal=sleep-interval=20
```

或者从部署文件创建ConfigMap:

```
# 或从文件创建
cat <<EOF > fortune-config.yml
apiVersion: v1
kind: ConfigMap
metadata:
  name: fortune-config
data:
  sleep-interval: "10"
EOF
```

```
# 创建ConfigMap
k create -f fortune-config.yml

# 查看ConfigMap的配置
k get cm fortune-config -o yaml
```

从ConfigMap获取配置数据,设置为pod的环境变量

```
cat <<EOF > fortune-pod-env-configmap.yml
apiVersion: v1
kind: Pod
metadata:
  name: fortune
  labels:
    app: fortune
spec:
  containers:

  - image: luksa/fortune:env
    env:
    - name: INTERVAL              # 环境变量名
      valueFrom:
        configMapKeyRef:          # 环境变量的值从ConfigMap获取
          name: fortune-config    # 使用的ConfigMap名称
          key: sleep-interval     # 用指定的键从ConfigMap取数据
    name: html-genrator
    volumeMounts:
    - name: html
      mountPath: /var/htdocs

  - image: nginx:alpine
    name: web-server
    volumeMounts:
    - name: html
      mountPath: /usr/share/nginx/html
      readOnly: true
    ports:
    - containerPort: 80
      protocol: TCP
```

```
  volumes:
  - name: html
    emptyDir: {}
EOF
```

config-map-->env-->arg

配置环境变量后,可以在启动参数中使用环境变量

```
cat <<EOF > fortune-pod-args.yml
apiVersion: v1
kind: Pod
metadata:
  name: fortune
  labels:
    app: fortune
spec:
  containers:

  - image: luksa/fortune:args
    env:
    - name: INTERVAL
      valueFrom:
        configMapKeyRef:
          name: fortune-config
          key: sleep-interval
    args: ["\$(INTERVAL)"]        # 启动参数中使用环境变量
    name: html-genrator
    volumeMounts:
    - name: html
      mountPath: /var/htdocs

  - image: nginx:alpine
    name: web-server
    volumeMounts:
    - name: html
      mountPath: /usr/share/nginx/html
      readOnly: true
    ports:
    - containerPort: 80
      protocol: TCP

  volumes:
  - name: html
    emptyDir: {}
EOF
```

从磁盘文件创建 ConfigMap 先删除之前创建的ComfigMap

```
d delete cm fortune-config
```

创建一个文件夹,存放配置文件

```
cd ~/
mkdir configmap-files
cd configmap
```

创建nginx的配置文件,启用对文本文件和xml文件的压缩

```
cat <<EOF > my-nginx-config.conf
server {
    listen           80;
    server_name      www.kubia-example.com;

    gzip             on;
    gzip_types       text/plain application/xml;

    location / {
        root         /ur/share/nginx/html;
        index        index.html index.htm;
    }
}
EOF
```

添加`sleep-interval`文件,写入值25

```
cat <<EOF > sleep-interval
25
EOF
```

从configmap-files文件夹创建ConfigMap

```
cd ~/

k create configmap fortune-config \
--from-file=configmap-files
```