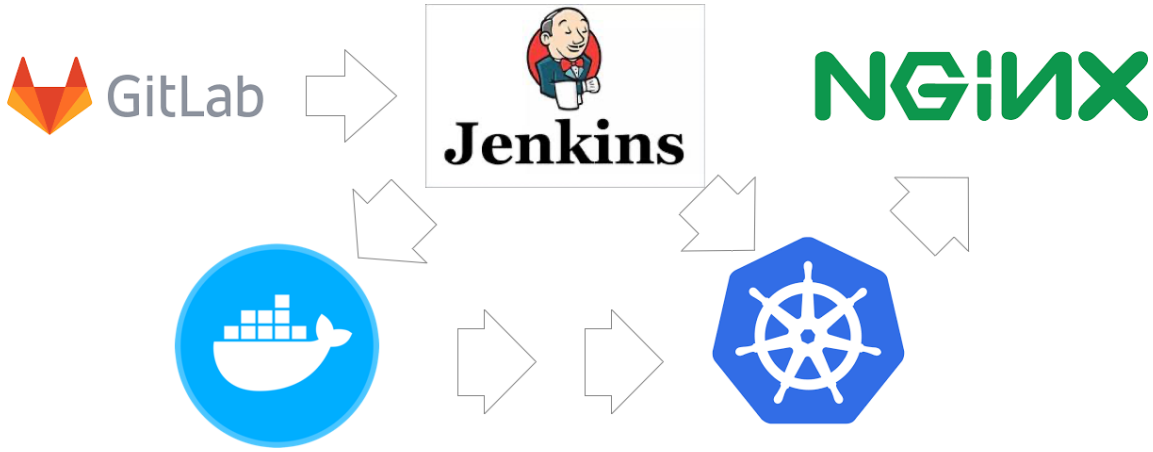
讲师(老司机)

docker&K8S直播



docker部分

直播间

```
上课地址：
https://view.csslcloud.net/api/view/lecturer?
roomid=F4B99D9C4B743D349C33DC5901307461&userid=22E1B7BD81D1E63A
讲师口令：123
```

服务器配置

硬件信息

序号	硬件信息	相关配置
1	操作系统	centos7.8
2	系统内核	4.4
3	系统用户	root
4	用户密码	123456
5	docker信息	19.03.12
6	docker-compose信息	1.25.5

节点信息

主机名	IP地址
mysql-110	192.168.198.110

上传下载

```
yum install -y lrzsz
```

数据库

基础镜像

拉取镜像

```
docker pull mysql:5.7.31
docker pull openjdk:8-alpine3.9
```

备份镜像

```
docker save mysql:5.7.31 -o mysql.5.7.31.tar
docker save openjdk:8-alpine3.9 -o openjdk8.tar
```

导入镜像

```
docker load -i mysql.5.7.31.tar
docker load -i openjdk8.tar

rm -rf *.tar
```

运行镜像

```
docker run -itd --name mysql --restart always -p 3306:3306 -e
MYSQL_ROOT_PASSWORD=admin mysql:5.7.31


docker logs -f mysql
```

测试连接

```
使用sqlLog客户端测试是否能正确连接mysql5.7
192.168.198.110
root
admin
```

创建数据库

```
创建lagou数据库
```

创建用户表

```
创建tbuser表
CREATE TABLE `tbuser` (
  `userid` int(11) NOT NULL AUTO_INCREMENT,
  `username` varchar(20) COLLATE utf8_bin DEFAULT NULL,
  `password` varchar(20) COLLATE utf8_bin DEFAULT NULL,
  `userroles` varchar(2) COLLATE utf8_bin DEFAULT NULL,
  `nickname` varchar(50) COLLATE utf8_bin DEFAULT NULL,
  PRIMARY KEY (`userid`)
) ENGINE=InnoDB AUTO_INCREMENT=3 DEFAULT CHARSET=utf8
COLLATE=utf8_bin
```

新增测试数据

```
INSERT INTO tbuser (username,PASSWORD,userroles,nickname) VALUES
('admin','1234','04','管理员'),('lagou','1234','03','拉勾教育')
```

导入数据库

导出本地数据库

推荐使用sqlLog客户端导出lagou.sql

查看初始化目录

在用docker创建mysql容器的时，有时候我们期望容器启动后数据库和表已经自动建好，初始化数据也已自动录入，也就是说容器启动后我们就能直接连上容器中的数据库，使用其中的数据了。我们一般都将sql文件放置在/docker-entrypoint-initdb.d目录中。

```
docker exec -it mysql bash

ls

cd /docker-entrypoint-initdb.d

exit
```

删除测试容器

```
docker stop mysql
docker rm mysql

mkdir -p /data/initsql
cd /data/initsql

将lagou.sql文件上传到/data/initsql目录中
```

Dockerfile

```
FROM mysql:5.7.31
# 作者信息
MAINTAINER mysql from date UTC by Asia/Shanghai
"laosiji@lagou.com"
ENV TZ Asia/Shanghai

COPY lagou.sql /docker-entrypoint-initdb.d
```

制作镜像

```
docker build --rm -t lagou/mysql:5.7 .
docker images

备份镜像，k8s作业中需要用到镜像
docker save lagou/mysql:5.7 -o lagou.mysql.5.7.tar

将镜像上传windows系统备份
sz lagou.mysql.5.7.tar
```

运行镜像

```
docker run -itd --name mysql --restart always -p 3306:3306 -e
MYSQL_ROOT_PASSWORD=admin lagou/mysql:5.7


docker logs -f mysql
```

sqlyog客户端测试

```
192.168.198.110
username:root
password:admin
```

删除测试容器

```
docker stop mysql
docker rm mysql
```

删除镜像

```
docker rmi -f lagou/mysql:5.7
```

springboot项目

项目简介

1. 使用springboot技术
2. mysql数据库
3. springboot项目docker容器化部署
4. mysql数据库容器化部署

本地测试项目

```
http://localhost:8082/users
```

打包项目

```
mvn clean package
```

制作镜像

打包项目

```
将jar包上传110节点/data/dockerdemo目录

mkdir -p /data/dockerdemo
cd /data/dockerdemo
```

Dockerfile

```
FROM openjdk:8-alpine3.9
# 作者信息
MAINTAINER laosiji Docker springboot "laosiji@lagou.com"
# 修改源
RUN echo "http://mirrors.aliyun.com/alpine/latest-stable/main/" >
/etc/apk/repositories && \
    echo "http://mirrors.aliyun.com/alpine/latest-
stable/community/" >> /etc/apk/repositories

# 安装需要的软件，解决时区问题
RUN apk --update add curl bash tzdata && \
    rm -rf /var/cache/apk/*

#修改镜像为东八区时间
ENV TZ Asia/Shanghai

COPY dockerdemo.jar app.jar
EXPOSE 8082
ENTRYPOINT ["java","-jar","/app.jar"]
```

制作镜像

```
docker build --rm -t lagou/dockerdemo:1.0 .

docker images
```

运行镜像

```
docker run -itd --name dockerdemo -p 8082:8082
lagou/dockerdemo:1.0

docker logs -f dockerdemo
```

测试项目

```
http://192.168.198.110:8082/users
```

k8s部分

基础镜像

拉取镜像

```
下载镜像：
    nfs动态存储
    docker pull vbouchaud/nfs-client-provisioner:v3.1.1
    测试nfs动态存储是否成功
    docker pull nginx:1.19.3-alpine
    dockerdemo项目自定义mysql镜像
    docker pull lagou/mysql:5.7
    微服项目需要的基础镜像
    docker pull openjdk:8-alpine3.9
```

导入镜像

master节点向集群每一个工作节点导入镜像信息。

```
master节点：
cd /data/

scp lagou.mysql.5.7.tar root@192.168.198.157:/data/
scp lagou.mysql.5.7.tar root@192.168.198.158:/data/
scp lagou.mysql.5.7.tar root@192.168.198.159:/data/

scp nfs-client-provisioner_v3.1.1.tar root@192.168.198.157:/data/
scp nfs-client-provisioner_v3.1.1.tar root@192.168.198.158:/data/
scp nfs-client-provisioner_v3.1.1.tar root@192.168.198.159:/data/

scp nginx.1.19.3.alpine.tar root@192.168.198.157:/data/
scp nginx.1.19.3.alpine.tar root@192.168.198.158:/data/
scp nginx.1.19.3.alpine.tar root@192.168.198.159:/data/

master节点单独导入
    docker load -i jdk8.tar

所有节点
    docker load -i lagou.mysql.5.7.tar
    docker load -i nfs-client-provisioner_v3.1.1.tar
    docker load -i nginx.1.19.3.alpine.tar
    rm -rf *.tar
    ls
```

服务器配置

硬件信息

序号	硬件信息	相关配置
1	操作系统	centos7.8
2	系统内核	4.4
3	系统用户	root
4	用户密码	123456
5	docker信息	19.03.12
6	docker-compose信息	1.25.5
7	k8s信息	1.17.5

节点信息

主机名	功能描述	IP地址
k8s-master-156	1.k8s集群master节点。 2.NFS服务-server端。	192.168.198.156
k8s-agent-157	工作节点。安装NFS服务。	192.168.198.157
k8s-agent-158	工作节点。安装NFS服务。	192.168.198.158
k8s-agent-159	工作节点。安装NFS服务。	192.168.198.159

NFS服务

安装NFS

因为网速原因，所有节点已经提前初始化安装NFS服务

```
yum install -y nfs-utils rpcbind


在master-156节点创建目录
mkdir -p /nfs
chmod 777 /nfs

更改归属组与用户
chown -R nfsnobody:nfsnobody /nfs
```

```
配置共享目录
echo "/nfs *(insecure,rw,sync,no_root_squash)" > /etc/exports

创建mysql共享目录
mkdir -p /nfs/mysql

所有节点设置启动服务
systemctl start rpcbind && systemctl start nfs

所有节点设置开启启动
systemctl enable rpcbind && systemctl enable nfs

master节点检查配置是否生效
exportfs

worker节点检查配置是否生效
showmount -e 192.168.198.156
```

测试NFS挂载

基础镜像

```
在157、158、159节点下载nginx镜像
docker pull nginx:1.19.3-alpine
```

nginx.yml

```
master-156节点创建目录：
mkdir -p /data/nginx
cd /data/nginx
```

nginx.yml文件清单如下：

```
apiVersion: v1
kind: Pod
metadata:
  name: vol-nfs
  namespace: default
spec:
  volumes:
    - name: html
      nfs:
```

```
        path: /nfs
        server: 192.168.198.156
  containers:
    - name: myapp
      image: nginx:1.19.3-alpine
      volumeMounts:
        - name: html
          mountPath: /usr/share/nginx/html/
```

测试pod

```
kubectl apply -f nginx.yml
kubectl get pods -o wide
curl 10.81.58.234

无法访问nginx首页，因为nfs共享目录中没有indxe.html页面。在nfs服务共享目录中新建index.html页面。
echo "hello lagouedu" > /nfs/index.html

需要等待片刻才能正常访问到index.html的页面信息
curl 10.81.58.234

kubectl delete -f nginx.yml
```

RBAC

```
master-156节点创建目录：
mkdir -p /data/mysql
cd /data/mysql
```

mysqlrbac.yml清单如下：

```
kind: ServiceAccount
apiVersion: v1
metadata:
  name: nfs-client-provisioner
---
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
```

```yaml
metadata:
  name: nfs-client-provisioner-runner
rules:
  - apiGroups: [""]
    resources: ["persistentvolumes"]
    verbs: ["get", "list", "watch", "create", "delete"]
  - apiGroups: [""]
    resources: ["persistentvolumeclaims"]
    verbs: ["get", "list", "watch", "update"]
  - apiGroups: ["storage.k8s.io"]
    resources: ["storageclasses"]
    verbs: ["get", "list", "watch"]
  - apiGroups: [""]
    resources: ["events"]
    verbs: ["create", "update", "patch"]
---
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: run-nfs-client-provisioner
subjects:
  - kind: ServiceAccount
    name: nfs-client-provisioner
    namespace: default      #替换成要部署NFS Provisioner的namespace
roleRef:
  kind: ClusterRole
  name: nfs-client-provisioner-runner
  apiGroup: rbac.authorization.k8s.io
---
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: leader-locking-nfs-client-provisioner
rules:
  - apiGroups: [""]
    resources: ["endpoints"]
    verbs: ["get", "list", "watch", "create", "update", "patch"]
---
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: leader-locking-nfs-client-provisioner
subjects:
  - kind: ServiceAccount
    name: nfs-client-provisioner
```

```
    namespace: default      #替换成要部署NFS Provisioner的namespace
roleRef:
  kind: Role
  name: leader-locking-nfs-client-provisioner
  apiGroup: rbac.authorization.k8s.io
```

storageClass

mysqlstorageclass.yml

```
kind: Deployment
apiVersion: apps/v1
metadata:
  name: nfs-client-provisioner
  labels:
    app: nfs-client-provisioner
spec:
  replicas: 1
  strategy:
    #设置升级策略为删除再创建(默认为滚动更新)
    type: Recreate
  selector:
    matchLabels:
      app: nfs-client-provisioner
  template:
    metadata:
      labels:
        app: nfs-client-provisioner
    spec:
      serviceAccountName: nfs-client-provisioner
      containers:
        - name: nfs-client-provisioner
          #由于quay.io仓库部分镜像国内无法下载，所以替换为其他镜像地址
          image: vbouchaud/nfs-client-provisioner:v3.1.1
          volumeMounts:
            - name: nfs-client-root
              mountPath: /persistentvolumes
          env:
            - name: PROVISIONER_NAME
              value: nfs-client     #--- nfs-provisioner的名称，以后
设置的storageclass要和这个保持一致
            - name: NFS_SERVER
```

```
              value: 192.168.198.156    #NFS服务器地址，与
volumes.nfs.servers保持一致
            - name: NFS_PATH
              value: /nfs/mysql             #NFS服务共享目录地址，与
volumes.nfs.path保持一致
      volumes:
        - name: nfs-client-root
          nfs:
            server: 192.168.198.156     #NFS服务器地址，与
spec.containers.env.value保持一致
            path: /nfs/mysql                #NFS服务器目录，与
spec.containers.env.value保持一致

---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: nfs-storage-mysql
  annotations:
    storageclass.kubernetes.io/is-default-class: "true" #设置为默认
的storageclass
provisioner: nfs-client                                 #动态卷分配
者名称，必须和创建的"provisioner"变量中设置的name一致
parameters:
  archiveOnDelete: "true"                               #设置
为"false"时删除PVC不会保留数据,"true"则保留数据
mountOptions:
  - hard                                                #指定为硬挂
载方式
  - nfsvers=4                                           #指定NFS版
本，这个需要根据 NFS Server 版本号设置
```

PVC

mysqlpvc.yml

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  # pvc名称
  name: mysqlbpvc
```

```
spec:
  # 使用的存储类
  storageClassName: nfs-storage-mysql
  # 读写权限
  accessModes:
    - ReadWriteOnce
  # 定义容量
  resources:
    requests:
      storage: 5Gi
```

services

mysqlservice.yml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mysql-deploy
  labels:
    app: mysql-deploy
spec:
  replicas: 1
  template:
    metadata:
      name: mysql-deploy
      labels:
        app: mysql-deploy
    spec:
      containers:
        - name: mysql-deploy
          image: lagou/mysql:5.7
          imagePullPolicy: IfNotPresent
          ports:
            - containerPort: 3306
          env:
            - name: MYSQL_ROOT_PASSWORD
              #这是mysqlroot用户的密码
              value: admin
            - name: TZ
              value: Asia/Shanghai
          args:
            - "--character-set-server=utf8mb4"
```

```
            - "--collation-server=utf8mb4_unicode_ci"
          volumeMounts:
            - mountPath: /var/lib/mysql #容器内的挂载目录
              name: volume-mysql
      restartPolicy: Always
      volumes:
        - name: volume-mysql
          persistentVolumeClaim:
            claimName: mysqlbpvc

  selector:
    matchLabels:
      app: mysql-deploy
---
apiVersion: v1
kind: Service
metadata:
  name: mysql-svc
spec:
  selector:
    app: mysql-deploy
  ports:
    - port: 3306
      targetPort: 3306
      nodePort: 30036
  type: NodePort
```

部署mysql

在master-156服务器进行如下操作：

1. 将mysql所有yml文件上传156节点
2. 部署mysql服务

```
cd /data
mkdir mysql
cd mysql

部署服务
kubectl apply -f .
```

```
删除服务
kubectl delete -f .

删除动态PV
kubectl get pv
kubectl edit pv pvc-c07f88b0-8595-4e5c-a61a-c2764ce75b27

将spec.claimRef属性下的所有内容全部删除
claimRef:
    apiVersion: v1
    kind: PersistentVolumeClaim
    name: test-pvc
    namespace: default
    resourceVersion: "162046"
    uid: 59fb2735-9681-426a-8805-8c94685a07e3

kubectl get pv
kubeclt delete pv pvc-c07f88b0-8595-4e5c-a61a-c2764ce75b27

删除/nfs/mysql/里的共享目录
```

导入数据库

使用sqlyog客户端软件查看lagou.sql数据库是否导入集群。

springboot项目

本地测试

本地测试，需要调整application.yml的url连接地址

```
url: jdbc:mysql://192.168.198.156:30036/lagou?
characterEncoding=utf-
8&useSSL=false&useTimezone=true&serverTimezone=GMT%2B8
```

浏览器测试

```
http://localhost:8082/users
```

部署K8S

将项目打包上传到K8S集群master节点。制作好镜像后分发到集群每一个节点

application.yml

将项目的application.yml文件url地址修改为部署的mysql服务名称，端口号为pod容器端口号。其余部分不用调整

```
url: jdbc:mysql://mysql-svc:3306/lagou?characterEncoding=utf-8&useSSL=false&useTimezone=true&serverTimezone=GMT%2B8
```

打包项目

```
mvn clean package

将jar包上传156节点
```

Dockerfile

```
mkdir -p /data/dockerdemo
cd /data/dockerdemo
```

Dockerfile文件清单如下：

```
FROM openjdk:8-alpine3.9
# 作者信息
MAINTAINER laosiji Docker springboot "laosiji@lagou.com"
# 修改源
RUN echo "http://mirrors.aliyun.com/alpine/latest-stable/main/" > /etc/apk/repositories && \
```

```
    echo "http://mirrors.aliyun.com/alpine/latest-
stable/community/" >> /etc/apk/repositories

# 安装需要的软件，解决时区问题
RUN apk --update add curl bash tzdata && \
    rm -rf /var/cache/apk/*

#修改镜像为东八区时间
ENV TZ Asia/Shanghai

COPY dockerdemo.jar app.jar
EXPOSE 8082
ENTRYPOINT ["java","-jar","/app.jar"]
```

制作镜像

```
docker build --rm -t lagou/dockerdemo:1.0 .
```

分发镜像

```
docker save lagou/dockerdemo:1.0 -o demo.tar

scp demo.tar root@192.168.198.157:/data/
scp demo.tar root@192.168.198.158:/data/
scp demo.tar root@192.168.198.159:/data/

157、158、159节点导入镜像
docker load -i demo.tar
```

部署项目

dockerdemo.yml文件清单如下：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: k8s-demo
  labels:
    app: k8s-demo
spec:
```

```
  replicas: 1
  template:
    metadata:
      name: k8s-demo
      labels:
        app: k8s-demo
    spec:
      containers:
        - name: k8s-demo
          image: lagou/dockerdemo:1.0
          imagePullPolicy: IfNotPresent
          ports:
            - containerPort: 8082
      restartPolicy: Always
  selector:
    matchLabels:
      app: k8s-demo
---
apiVersion: v1
kind: Service
metadata:
  name: lagou-svc
spec:
  selector:
    app: k8s-demo
  ports:
    - port: 8082
      targetPort: 8082
      nodePort: 30082
  type: NodePort
```

运行服务

```
kubectl apply -f dockerdemo.yml
```

测试项目

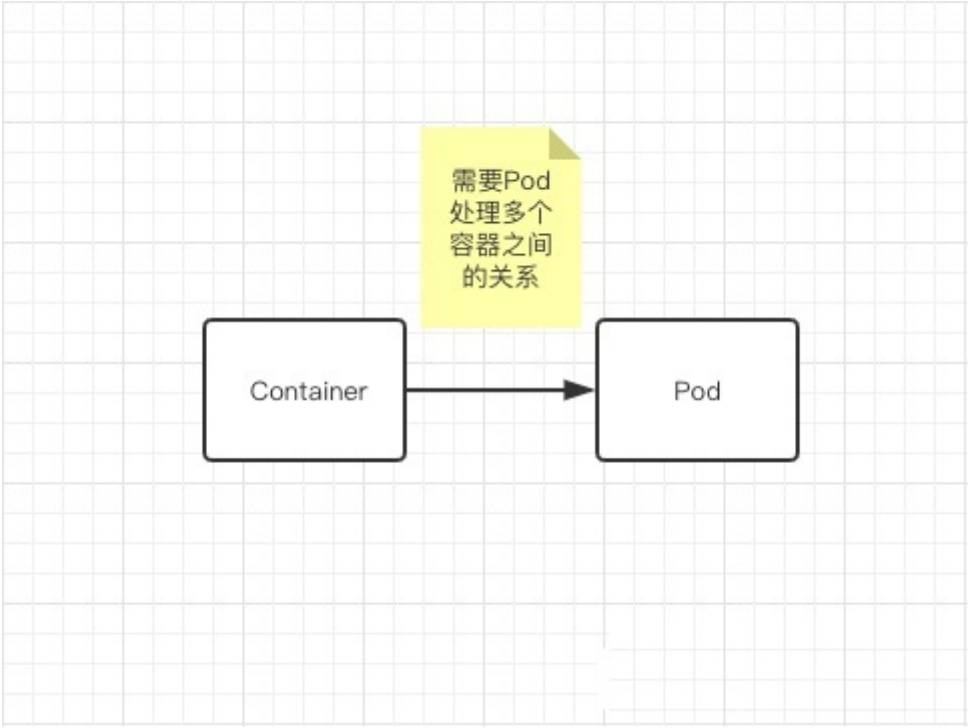
```
http://192.168.198.156:30082/users
```

答疑

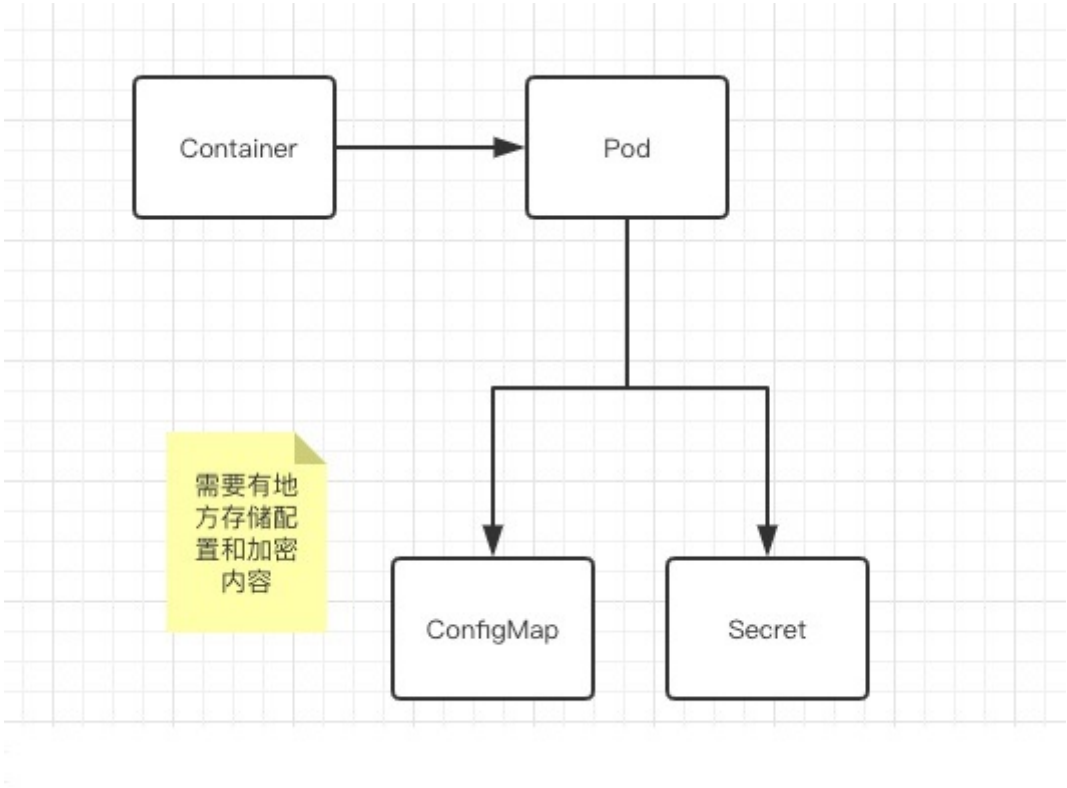
问题一：

pod和service的关系，以及service是如何负载均衡到对应的pod的?

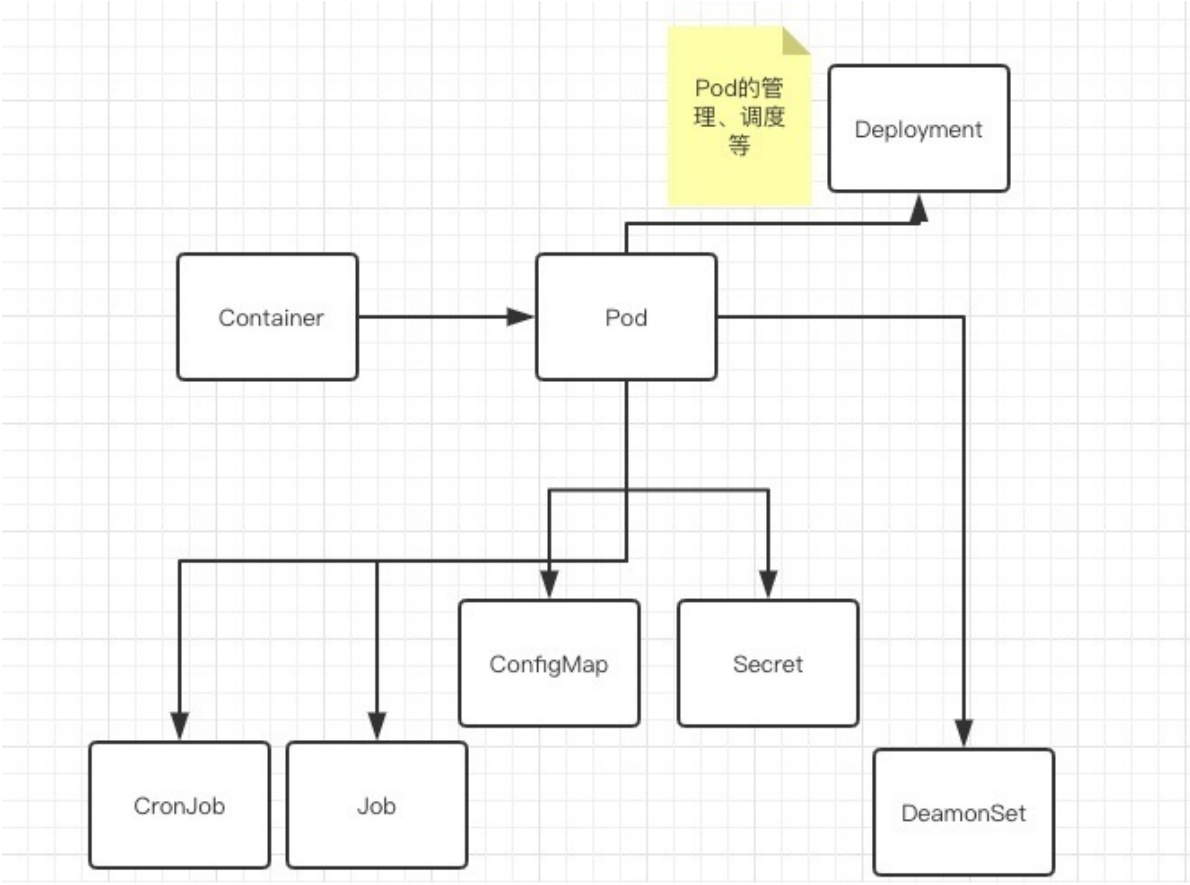
在学习k8s之前，首先学习是docker。我们`pull`一个镜像之后，要把应用运行起来最简单的方式使用**docker run**命令。然而在实际的生产环境中，很少仅靠一个单容器就能够满足。比如，一个web前端的应用，可能还得依赖后端的一个容器服务；后端的容器可能需要数据库服务；后端的服务需要多副本等等场景。在这些假想的场景中，比较真实的需求就是这些容器应用需要共享同一个网络，同一个存储卷等，还有它们的生命周期如何管理调度。这个时候，仅仅依靠容器无法解决这个问题，k8s集群的第一个选手**pod**就闪亮登场了。



有了pod之后，同一个pod内的容器可以共享很多信息，也可能需要读取同一份配置。比如pod内有两个容器需要访问同一个数据库，那么我们可以把相关的配置信息写到configMap里。那如果还有一些比较敏感的信息，就需要放到secret对象中。k8s 就会在你指定的 pod（比如，mysql应用的pod）启动时，自动把 secret 里的数据以 volume 的方式挂载到容器里。



下一个重点是deployment。前面讲过容器之间的关联关系、共享资源等问题需要处理，从而引入了pod。对于pod，也是同样的问题需要解决，只不过高了一个抽象层次罢了。因为面临pod的生命周期管理、调度、多副本等问题需要解决，k8s集群聪明的设计者引入了deployment控制器。它可以根据我们的需求（比如通过标签）将pod调度到目标机器上，调度完成之后，它还会继续帮我们继续监控容器是否在正确运行，一旦出现问题，会立刻告诉我们pod的运行不正常以及寻找可能的解决方案，比如目标节点不可用的时候它可以快速地调度到别的机器上去。另外，如果需要对应用扩容提升响应能力的时候，通过deployment可以快速地进行扩展。简单的理解，deployment帮助k8s管理pod。在我们学习过程中，deployment并不是直接控制着pod的，中间实际上还有一个replicaSet控制器。



提供容器服务

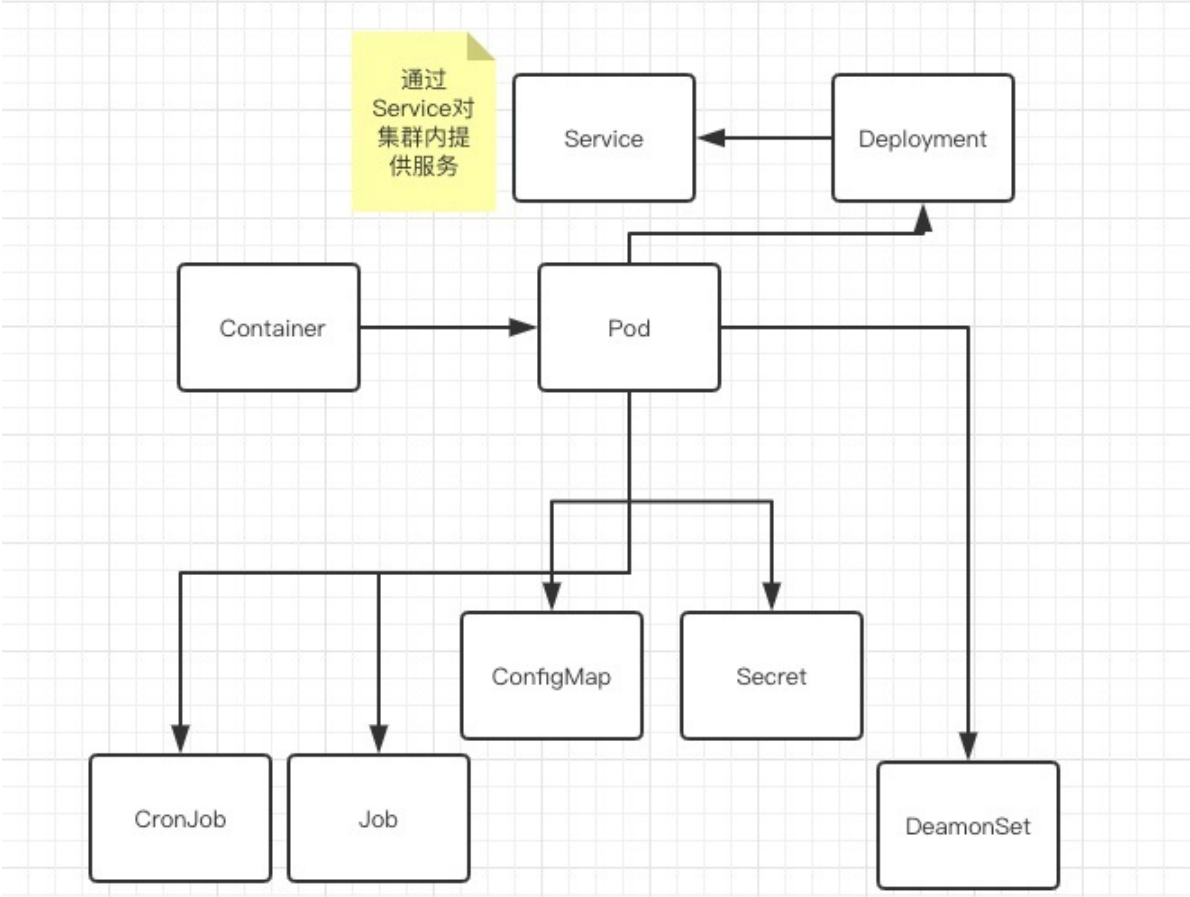
前面的内容是围绕着pod自身的运行调度管理，接下来面临的问题是解决如何将服务提供给第三方的问题。

对内提供服务

例如**8s作业中的mysql的服务`mysql-svc`，提供给dockerdemo的服务用。**

首先要解决的是将服务提供给同一个集群内的其他服务使用。有的同学问为什么我们不能直接使用pod的IP呢？前面介绍的pod是会被管理调度的，可能被调度到不同的机器上，同时生命周期也可能会发生变化。这导致一个应用pod的IP可能会随时发生变化，那么直接使用pod的IP自然是不合理的。针对这个问题K8s提供了service资源来解决。但是，并不是说service就有一个固定的IP。而且，它和pod IP还有很不一样的地方。pod的网络是k8s在物理机上建立了一层Overlay Network实现的，而且在网卡上能够看到这个网络的地址。但是service是一个完全虚拟的网络层，并不会存在于任何网络设备上。它通过修改集群内部的路由规则，仅对集群内部有效。deploment创建好应用之后，再为它生成一个service对象。接下来就可以通过service的域名访问到服务，形式是.，比如你有为deployment的应用创建了一个名为`mysql-svc`的service在默认的命名空间，那么集群内想要通过http访问这个应

用，就可以使用http://mysql-svc.default。这个域名仅在集群内有效，因为是k8s内部的一个DNS负责解析。

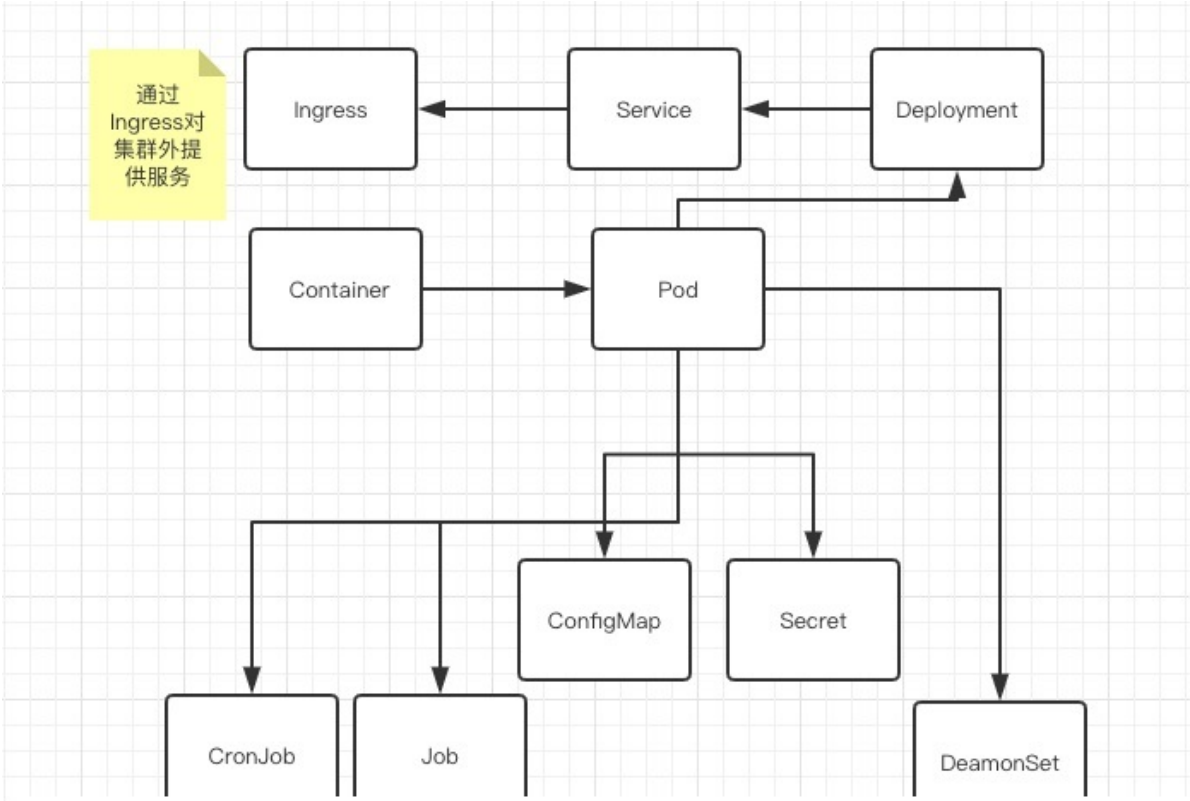


对外提供服务

例如**dockerdemo服务提供对外访问**

- port-forward 映射服务到端口
- nodeport 映射服务到节点端口
- 使用 loadblancer service 云平台暴露服务
- 通过 ingress 暴露服务

讲完如何给内部提供服务以后，剩下的就是如何给外部提供服务了。在k8s里推荐使用ingress，正如其名，它是集群的入口。比如我们的集群`dockerdemo`应用想要让用户能够访问，在实际工作中推荐使用ingress。事实上nginx ingress也是K8s生态中的成员。



问题二：

能否`详细讲一下 k8s做注册中心 的基本原理，如何使用`?请老师根据微服务课程应颠老师关于拉勾微服务架构图。简单介绍一下怎么用k8s进行这些微服务的部署？比如说哪些微服务可以放在同一个POD里？画一个简单的部署图?

1.不推荐把多个微服部署在同一个pod中。比如项目中的某一个微服(注册中心、网关、热点等)需要扩容或者缩容，如果我们把两个微服项目部署在同一个pod中。会造成多个微服一起扩容或者缩容。

2.微服项目理论上都可以部署到k8s集群中。但业界内有一部分专门做数据安全性比较高大咖认为mysql等关系型数据库部署到容器中会出现性能瓶颈等。

参考文章：今日头条中的一篇文章

```
https://www.toutiao.com/i6805798581971190276/
```

3.如何使用k8s部署微服项目

3.1开发人员方式：

3.1.1每一个微服项目进行打包+Dockerfile

3.1.2编写docker-compose.yml文件在docker中测试整个微服项目

3.1.3测试无误后交给运维工程师进行测试

3.1.4运维工程师使用专业部署工具进行部署

3.2运维人员方式：

专业的部署k8s集群工具。

3.2.1rancher系列

3.2.2kubesphere系列

4.部署顺序

4.1数据相关：

①关系型数据库集群:例如mysql集群

②非关系型数据库集群:例如redis集群

③搜索引擎集群:例如ELK集群

④图片处理集群:例如fastDFS集群

⑤部署微服项目:例如注册中心、网关等

⑥部署前端项目:vue等

问题三： 2虚拟机 k8s-mater k8s-work