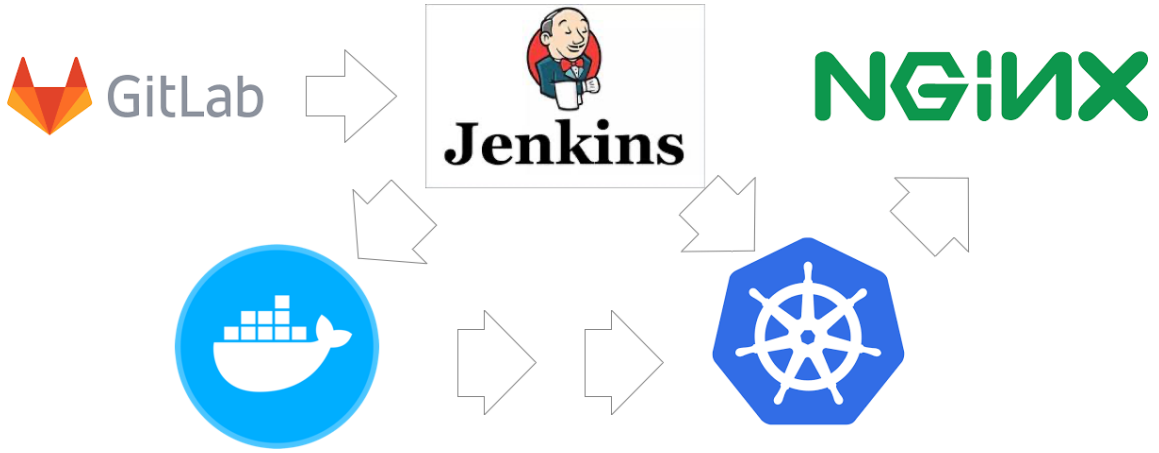
讲师(老司机)

容器虚拟化技术和自动化部署



k8s高级篇-volume(存储)

准备镜像

k8s集群每个node节点需要下载镜像：

```
docker pull mariadb:10.5.2
```

安装mariaDB

部署service

maria/mariadb.yml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mariadb
  labels:
    app: mariadb
spec:
  replicas: 1
  template:
    metadata:
      name: mariadb
      labels:
        app: mariadb
    spec:
      containers:
```

```
        - name: mariadb
          image: mariadb:10.5.2
          imagePullPolicy: IfNotPresent
          env:
            - name: MYSQL_ROOT_PASSWORD
              value: admin
            - name: TZ
              value: Asia/Shanghai
          args:
            - "--character-set-server=utf8mb4"
            - "--collation-server=utf8mb4_unicode_ci"
          ports:
            - containerPort: 3306
      restartPolicy: Always
  selector:
    matchLabels:
      app: mariadb
---
apiVersion: v1
kind: Service
metadata:
  name: mariadb-svc
spec:
  selector:
    app: mariadb
  ports:
    - port: 3306
      targetPort: 3306
      nodePort: 30036
  type: NodePort
```

运行服务

```
kubectl apply -f .

kubectl get pod -o wide
```

客户端测试

```
IP:192.168.198.157
username:root
password:admin
prot: 30036
```

删除service

```


kubectl delete -f mariadb.yml
```

secret

Secret 解决了密码、token、密钥等敏感数据的配置问题，而不需要把这些敏感数据暴露到镜像或者 Pod Spec中。Secret 可以以 Volume 或者环境变量的方式使用 。

Secret 有三种类型：

- Service Account： 用来访问 Kubernetes API，由 Kubernetes 自动创建，并且会自动挂载到 Pod 的
  /run/secrets/kubernetes.io/serviceaccount 目录中。
- Opaque： base64编码格式的Secret，用来存储密码、密钥等
- kubernetes.io/dockerconfigjson： 用来存储私有 docker registry 的认证信息

Service Account

Service Account简称sa， Service Account 用来访问 Kubernetes API，由 Kubernetes 自动创建，并且会自动挂载到 Pod的/run/secrets/kubernetes.io/serviceaccount 目录中。

```
查询命名空间为kube-system的pod信息
kubectl get pod -n kube-system

进入pod:kube-proxy-48bz4
kubectl exec -it kube-proxy-48bz4 -n kube-system sh

cd /run/secrets/kubernetes.io/serviceaccount
ls

cat ca.crt
cat namespace
cat token
```

Opaque Secret

Opaque 类型的数据是一个 map 类型，要求 value 是 base64 编码格式。

加密、解密

使用命令行方式对需要加密的字符串进行加密。例如:mysql数据库的密码。

```
对admin字符串进行base64加密:获得admin的加密字符串"YWRtaW4="
echo -n "admin" | base64



base64解密：对加密后的字符串进行解密
echo -n "YWRtaW4=" | base64 -d

```

资源文件方式创建

对mariadb数据库密码进行加密

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: mariadbsecret
type: Opaque
data:
  password: YWRtaW4=
  #mariadb的用户名root加密，用于演示，无实际效果
  username: cm9vdA==
```

升级mariadb的service

```yaml
        env:
            - name: MYSQL_ROOT_PASSWORD
              valueFrom:
                secretKeyRef:
                  key: password
                  name: mariadbsecret
            - name: TZ
              value: Asia/Shanghai
```

全部资源文件清单

secret/mariadbsecret.yml

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: mariadbsecret
type: Opaque
data:
  password: YWRtaW4=
  #mariadb的用户名root加密，用于演示，无实际效果
  username: cm9vdA==
```

secret/mariadb.yml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mariadb
  labels:
    app: mariadb
spec:
  replicas: 1
```

```yaml
  template:
    metadata:
      name: mariadb
      labels:
        app: mariadb
    spec:
      containers:
        - name: mariadb
          image: mariadb:10.5.2
          imagePullPolicy: IfNotPresent
          env:
            - name: MYSQL_ROOT_PASSWORD
              valueFrom:
                secretKeyRef:
                  key: password
                  name: mariadbsecret
            - name: TZ
              value: Asia/Shanghai
          args:
            - "--character-set-server=utf8mb4"
            - "--collation-server=utf8mb4_unicode_ci"
          ports:
            - containerPort: 3306
      restartPolicy: Always
  selector:
    matchLabels:
      app: mariadb
---
apiVersion: v1
kind: Service
metadata:
  name: mariadb-svc
spec:
  selector:
    app: mariadb
  ports:
    - port: 3306
      targetPort: 3306
      nodePort: 30036
  type: NodePort
```

运行service

```
kubectl apply -f .

kubectl get secret
kubectl get svc
```

客户端测试

```
IP:192.168.198.157
username:root
password:admin
prot: 30036
```

删除service、secret

```
kubectl delete -f .

kubectl get secret
kubectl get svc
```

安装harbor私服

harbor官网地址:

```
harbor官网地址:
https://goharbor.io/

github官网地址:
https://github.com/goharbor/harbor
```

docker-compose

```
验证docker-compose
docker-compose -v
```

安装harbor

```
1.解压软件
cd /data
tar zxf harbor-offline-installer-v1.9.4.tgz

2.进入安装目录
cd harbor

3.修改配置文件
vi harbor.yml
3.1修改私服镜像地址
hostname: 192.168.198.155
3.2修改镜像地址访问端口号
port: 5000
3.3harbor管理员登录系统密码
harbor_admin_password: Harbor12345
```

```
3.4修改harbor映射卷目录
data_volume: /data/harbor


4.安装harbor
4.1执行启动脚本,经过下述3个步骤后，成功安装harbor私服
./install.sh
4.2准备安装环境：检查docker版本和docker-compose版本
4.3加载harbor需要的镜像
4.4准备编译环境
4.5启动harbor。通过docker-compose方式启动服务
4.6google浏览器访问harbor私服
http://192.168.198.155:5000
    username: admin
    password: Harbor12345
```

新建项目

```
在harbor中新建公共项目：
laogouedu
```

配置私服

```
k8s集群master节点配置docker私服：master节点用于上传镜像。其余工作节点暂时不要配置私服地址。

vi /etc/docker/daemon.json
"insecure-registries":["192.168.198.155:5000"]

重启docker服务：
systemctl daemon-reload
systemctl restart docker
```

登录私服

```
docker login -u admin -p Harbor12345 192.168.198.155:5000


退出私服
docker logout 192.168.198.155:5000
```

上传mariadb镜像

```
docker tag mariadb:10.5.2 192.168.198.155:5000/lagouedu/mariadb:10.5.2

docker push 192.168.198.155:5000/lagouedu/mariadb:10.5.2

docker rmi -f 192.168.198.155:5000/lagouedu/mariadb:10.5.2
```

修改mariadb镜像地址

修改secret/mariadb.yml文件，将image地址修改为harbor私服地址

```
 image: 192.168.198.155:5000/lagouedu/mariadb:10.5.2
```

运行服务

```
kubectl apply -f .

查看pod信息：发现镜像拉取失败，STATUS显示信息为"ImagePullBackOff"
kubectl get pods

查看pod详细信息:拉取harbor私服镜像失败。
kubectl describe pod mariadb-7b6f895b5b-mc5xp


删除服务：
kubectl delete -f .
```

注册私服

使用 Kuberctl 创建 docker registry 认证的 secret

```
语法规则：
kubectl create secret docker-registry myregistrykey --docker-server=REGISTRY_SERVER --docker-username=DOCKER_USER --docker-password=DOCKER_PASSWORD --docker-email=DOCKER_EMAIL

例子：
kubectl create secret docker-registry lagouharbor --docker-server=192.168.198.155:5000 --docker-username=admin --docker-password=Harbor12345 --docker-email=harbor@lagou.com


k8s集群其余工作节点配置docker私服地址：
vi /etc/docker/daemon.json
"insecure-registries":["192.168.198.155:5000"]

```

```
重启docker服务：
systemctl daemon-reload
systemctl restart docker
```

secret升级mariadb

将mariadb镜像修改为harbor私服地址。 在创建 Pod 的时候，通过 imagePullSecrets 来引用刚创建的 `myregistrykey`

```
spec:
    imagePullSecrets:
        - name: lagouharbor
    containers:
        - name: mariadb
        image: 192.168.198.155:5000/lagouedu/mariadb:10.5.2

```

全部资源文件清单

mariadbsecret.yml

```
apiVersion: v1
kind: Secret
metadata:
  name: mariadbsecret
type: Opaque
data:
  password: YWRtaW4=
  #mariadb的用户名root加密，用于演示，无实际效果
  username: cm9vdA==
```

mariadb.yml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mariadb
  labels:
    app: mariadb
spec:
  replicas: 1
  template:
    metadata:
      name: mariadb
      labels:
        app: mariadb
    spec:
      imagePullSecrets:
        - name: lagouharbor
```

```yaml
      containers:
        - name: mariadb
          image: 192.168.198.155:5000/lagouedu/mariadb:10.5.2
          imagePullPolicy: IfNotPresent
          env:
            - name: MYSQL_ROOT_PASSWORD
              valueFrom:
                secretKeyRef:
                  key: password
                  name: mariadbsecret
            - name: TZ
              value: Asia/Shanghai
          args:
            - "--character-set-server=utf8mb4"
            - "--collation-server=utf8mb4_unicode_ci"
          ports:
            - containerPort: 3306
      restartPolicy: Always
  selector:
    matchLabels:
      app: mariadb
---
apiVersion: v1
kind: Service
metadata:
  name: mariadb-svc
spec:
  selector:
    app: mariadb
  ports:
    - port: 3306
      targetPort: 3306
      nodePort: 30036
  type: NodePort
```

客户端测试

```
IP:192.168.198.157
username:root
password:admin
prot: 30036
```

configmap

ConfigMap顾名思义，是用于保存配置数据的键值对，可以用来保存单个属性，也可以保存配置文件。

ConfigMaps允许你将配置构件与映像内容解耦，以保持容器化应用程序的可移植性。configmap 可以从文件、目录或者 key-value 字符串创建等创建 ConfigMap。也可以通过 kubectl create -f从描述文件创建。可以使用 kubectl create创建命令。创建ConfigMap的方式有4种：

1. 直接在命令行中指定configmap参数创建，即--from-literal。
2. 指定文件创建，即将一个配置文件创建为一个ConfigMap--from-file=<文件>
3. 指定目录创建，即将一个目录下的所有配置文件创建为一个ConfigMap，--from-file=<目录>
4. 先写好标准的configmap的yaml文件，然后kubectl create -f 创建

命令行方式

从key-value字符串创建，官方翻译是从字面值中创建ConfigMap。

语法规则

```
kubectl create configmap map的名字 --from-literal=key=value

如果有多个key，value。可以继续在后边写
kubectl create configmap map的名字
    --from-literal=key1=value1
    --from-literal=key2=value2
    --from-literal=key3=value4
```

案例

```
创建configmap
kubectl create configmap helloconfigmap --from-literal=lagou.hello=world

查看configmap
kubectl get configmap helloconfigmap -o go-template='{{.data}}'
```

```
在使用kubectl get获取资源信息的时候,可以通过-o(--output简写形式)指定信息输出的格式,如果指定的是yaml或者json输出的是资源的完整信息,实际工作中,输出内容过少则得不到我们想要的信息,输出内容过于详细又不利于快速定位的我们想要找到的内容,其实-o输出格式可以指定为go-template然后指定一个template,这样我们就可以通过go-template获取我们想要的内容.go-template与kubernetes无关,它是go语言内置的一种模板引擎.这里不对go-template做过多解释,仅介绍在kubernetes中获取资源常用的语法,想要获取更多内容,大家可以参考相关资料获取帮助。大家记住是固定语法即可。
```

配置文件方式

语法规则

```
语法规则如下：当 --from-file指向一文件，key的名称是文件名称，value的值是这个文件的内容。

kubectl create configmap cumulx-test --from-file=xxxx
```

案例

创建一个配置文件:jdbc.properties

```
vi jdbc.properties

jdbc.driverclass=com.mysql.jdbc.Driver
jdbc.url=jdbc:mysql://localhost:3306/test
jdbc.username=root
jdbc.password=admin
```

```
创建configmap换用其他方式查看信息
kubectl create configmap myjdbcmap  --from-file=jdbc.properties

查看configmap详细信息
kubectl describe configmaps myjdbcmap
```

目录方式

语法规则

```
语法规则如下：当 --from-file指向一个目录，每个目录中的文件直接用于填充ConfigMap中的
key,key的名称是文件名称，value的值是这个文件的内容。下面的命令读取/data目录下的所有文件
kubectl create configmap cumulx-test --from-file=/data/
```

案例

```
kubectl create configmap myjdbcconfigmap --from-file=/data/jdbc.properties

查看configmap详细信息
kubectl describe configmaps myjdbcmap
```

资源文件方式

yml文件

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: mariadbconfigmap
data:
  mysql-driver: com.mysql.jdbc.Driver
  mysql-url: jdbc:mysql://localhost:3306/test
  mysql-user: root
  mysql-password: admin
```

运行yml文件

```
kubectl apply -f mariadb-configmap.yml

查看configmap详细信息
kubectl describe configmaps mariadbconfigmap
```

configmap升级mariadb

获得my.cnf文件

```
运行测试镜像：
docker run --name some-mariadb -e MYSQL_ROOT_PASSWORD=my-secret-pw -d
192.168.198.155:5000/lagouedu/mariadb:10.5.2

将my.cnf文件从容器中复制到/data目录
docker cp some-mariadb:/etc/mysql/my.cnf /data

停止容器
docker stop some-mariadb

删除容器
docker rm some-mariadb

删除镜像
docker rmi -f 192.168.198.155:5000/lagouedu/mariadb:10.5.2

更改my.cnf文件中的端口号为：3307
```

生成configmap文件

data/mariadbconfigmap.yml

```
将my.cnf文件中mariadb数据库启动端口号修改为3307，将my.cnf文件上传master节点，首先通过命令行方式生成configmap资源。再通过命令行方式将configmap反向生成yaml文件。将生成文件copy回idea开发工具。修改yaml文件备用。

kubectl create configmap mysqlini --from-file=my.cnf
kubectl get configmap mysqlini -o yaml > mariadbconfigmap.yml
```

修改mariadb.yml

```

        volumeMounts:
            - mountPath: /etc/mysql/mariadb.conf.d/
              name: mariadbconfigmap
      restartPolicy: Always
      volumes:
        - name: mariadbconfigmap
          configMap:
            name: mariadbconfigmap
```

全部资源文件清单

configmap/mariadbsecret.yml

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: mariadbsecret
type: Opaque
data:
  password: YWRtaW4=
  #mariadb的用户名root加密，用于演示，无实际效果
  username: cm9vdA==

```

configmap/mariadb.yml--需要再次整理，加入私服镜像信息

注意修改pod的端口号为3307，service的targetPort端口号为3307

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mariadb-deploy
  labels:
    app: mariadb-deploy
spec:
  replicas: 1
  template:
    metadata:
```

```yaml
      name: mariadb-deploy
      labels:
        app: mariadb-deploy
    spec:
      imagePullSecrets:
        - name: lagouharbor
      containers:
        - name: mariadb-deploy
          image: 192.168.198.155:5000/lagouedu/mariadb:10.5.2
          imagePullPolicy: IfNotPresent
          ports:
            - containerPort: 3307
          env:
            - name: MYSQL_ROOT_PASSWORD
              #这是mysqlroot用户的密码
              valueFrom:
                secretKeyRef:
                  key: password
                  name: mariadbsecret
            - name: TZ
              value: Asia/Shanghai
          args:
            - "--character-set-server=utf8mb4"
            - "--collation-server=utf8mb4_unicode_ci"
          volumeMounts:
            - mountPath: /etc/mysql/mariadb.conf.d/   #容器内的挂载目录
              name: lagoumariadb #随便给一个名字,这个名字必须与volumes.name一致
      restartPolicy: Always
      volumes:
        - name: lagoumariadb
          configMap:
            name: mariadbconfigmap
  selector:
    matchLabels:
      app: mariadb-deploy
---
apiVersion: v1
kind: Service
metadata:
  name: mariadb-svc
spec:
  selector:
    app: mariadb-deploy
  ports:
    - port: 3307
      targetPort: 3307
      nodePort: 30036
  type: NodePort
```

configmap/mariadbconfigmap.yml

```
apiVersion: v1
data:
  my.cnf: "# MariaDB database server configuration file.\n#\n# You can copy this file
    to one of:\n# - \"/etc/mysql/my.cnf\" to set global options,\n# - \"~/.my.cnf\"
    to set user-specific options.\n# \n# One can use all long options that the program
    supports.\n# Run program with --help to get a list of available options and with\n#
    --print-defaults to see which it would actually understand and use.\n#\n# For
    explanations see\n# http://dev.mysql.com/doc/mysql/en/server-system-variables.html\n\n#
    This will be passed to all mysql clients\n# It has been reported that passwords
    should be enclosed with ticks/quotes\n# escpecially if they contain \"#\" chars...\n#
    Remember to edit /etc/mysql/debian.cnf when changing the socket location.\n[client]\nport\t\t=
    3307\nsocket\t\t= /var/run/mysqld/mysqld.sock\n\n# Here is entries for some specific
    programs\n# The following values assume you have at least 32M ram\n\n# This was
    formally known as [safe_mysqld]. Both versions are currently parsed.\n[mysqld_safe]\nsocket\t\t=
    /var/run/mysqld/mysqld.sock\nnice\t\t= 0\n\n[mysqld]\n#\n# * Basic Settings\n#\n#user\t\t=
    mysql\npid-file\t= /var/run/mysqld/mysqld.pid\nsocket\t\t= /var/run/mysqld/mysqld.sock\nport\t\t=
    3307\nbasedir\t\t= /usr\ndatadir\t\t= /var/lib/mysql\ntmpdir\t\t= /tmp\nlc_messages_dir\t=
    /usr/share/mysql\nlc_messages\t= en_US\nskip-external-locking\n#\n# Instead of
    skip-networking the default is now to listen only on\n# localhost which is more
    compatible and is not less secure.\n#bind-address\t\t= 127.0.0.1\n#\n# * Fine
    Tuning\n#\nmax_connections\t\t= 100\nconnect_timeout\t\t= 5\nwait_timeout\t\t=
    600\nmax_allowed_packet\t= 16M\nthread_cache_size       = 128\nsort_buffer_size\t=
    4M\nbulk_insert_buffer_size\t= 16M\ntmp_table_size\t\t= 32M\nmax_heap_table_size\t=
    32M\n#\n# * MyISAM\n#\n# This replaces the startup script and checks MyISAM tables
    if needed\n# the first time they are touched. On error, make copy and try a repair.\nmyisam_recover_options
    = BACKUP\nkey_buffer_size\t\t= 128M\n#open-files-limit\t= 2000\ntable_open_cache\t=
    400\nmyisam_sort_buffer_size\t= 512M\nconcurrent_insert\t= 2\nread_buffer_size\t=
    2M\nread_rnd_buffer_size\t= 1M\n#\n# * Query Cache Configuration\n#\n# Cache only
```

```
    tiny result sets, so we can fit more in the query
cache.\nquery_cache_limit\t\t=
    128K\nquery_cache_size\t\t= 64M\n# for more write intensive setups, set
to DEMAND
    or OFF\n#query_cache_type\t\t= DEMAND\n#\n# * Logging and
Replication\n#\n# Both
    location gets rotated by the cronjob.\n# Be aware that this log type is
a performance
    killer.\n# As of 5.1 you can enable the log at
runtime!\n#general_log_file        =
    /var/log/mysql/mysql.log\n#general_log             = 1\n#\n# Error
logging goes
    to syslog due to /etc/mysql/conf.d/mysqld_safe_syslog.cnf.\n#\n# we do
want to
    know about network errors and such\n#log_warnings\t\t= 2\n#\n# Enable
the slow
    query log to see queries with especially long
duration\n#slow_query_log[={0|1}]\nslow_query_log_file\t=
    /var/log/mysql/mariadb-slow.log\nlong_query_time =
10\n#log_slow_rate_limit\t=
    1000\n#log_slow_verbosity\t= query_plan\n\n#log-queries-not-using-
indexes\n#log_slow_admin_statements\n#\n#
    The following can be used as easy to replay backup logs or for
replication.\n#
    note: if you are setting up a replication slave, see README.Debian
about\n#       other
    settings you may need to change.\n#server-id\t\t= 1\n#report_host\t\t=
master1\n#auto_increment_increment
    = 2\n#auto_increment_offset\t= 1\n#log_bin\t\t\t=
/var/log/mysql/mariadb-bin\n#log_bin_index\t\t=
    /var/log/mysql/mariadb-bin.index\n# not fab for performance, but
safer\n#sync_binlog\t\t=
    1\nexpire_logs_days\t= 10\nmax_binlog_size         = 100M\n#
slaves\n#relay_log\t\t=
    /var/log/mysql/relay-bin\n#relay_log_index\t= /var/log/mysql/relay-
bin.index\n#relay_log_info_file\t=
    /var/log/mysql/relay-bin.info\n#log_slave_updates\n#read_only\n#\n# If
applications
    support it, this stricter sql_mode prevents some\n# mistakes like
inserting invalid
    dates etc.\n#sql_mode\t\t= NO_ENGINE_SUBSTITUTION,TRADITIONAL\n#\n# *
InnoDB\n#\n#
    InnoDB is enabled by default with a 10MB datafile in /var/lib/mysql/.\n#
Read
    the manual for more InnoDB related options. There are
many!\ndefault_storage_engine\t=
    InnoDB\ninnodb_buffer_pool_size\t= 256M\ninnodb_log_buffer_size\t=
8M\ninnodb_file_per_table\t=
    1\ninnodb_open_files\t= 400\ninnodb_io_capacity\t=
400\ninnodb_flush_method\t=
    O_DIRECT\n#\n# * Security Features\n#\n# Read the manual, too, if you
want chroot!\n#
    chroot = /var/lib/mysql/\n#\n# For generating SSL certificates I
recommend the
    OpenSSL GUI \"tinyca\".\n#\n# ssl-ca=/etc/mysql/cacert.pem\n# ssl-
cert=/etc/mysql/server-cert.pem\n#
    ssl-key=/etc/mysql/server-key.pem\n\n#\n# * Galera-related
settings\n#\n[galera]\n#
```

```
    Mandatory settings\n#wsrep_on=ON\n#wsrep_provider=\n#wsrep_cluster_address=\n#binlog_format=row\n#default_storage_engine=InnoDB\n#innodb_autoinc_lock_mode=2\n#\n#
    Allow server to accept connections on all interfaces.\n#\n#bind-address=0.0.0.0\n#\n#
    Optional setting\n#wsrep_slave_threads=1\n#innodb_flush_log_at_trx_commit=0\n\n[mysqldump]\nquick\nquote-names\nmax_allowed_packet\t=
    16M\n\n[mysql]\n#no-auto-rehash\t# faster start of mysql but no tab completion\n\n[isamchk]\nkey_buffer\t\t=
    16M\n\n#\n# * IMPORTANT: Additional settings that can override those from this
    file!\n#   The files must end with '.cnf', otherwise they'll be ignored.\n#\n!include
    /etc/mysql/mariadb.cnf\n!includedir /etc/mysql/conf.d/\n"
kind: ConfigMap
metadata:
  name: mariadbconfigmap
  namespace: default
```

运行服务

```
部署服务
kubectl apply -f .

查看相关信息
kubectl get configmaps
kubectl get pods
```

查看mariadb容器日志

```
查看mariadb容器日志：观察mariadb启动port是否为3307
kubectl logs -f mariadb-5d99587d4b-xwthc
```

客户端测试

```
IP:192.168.198.157
username:root
password:admin
prot: 30036
```

label操作

前面的课程我们学习了如何给pod打标签及修改pod的标签值。在某些特殊情况下，需要将某些服务固定在一台宿主机上, k8s可以使用label给node节点打上标签来满足这种需求。

添加label语法

```
kubectl label nodes <node-name> <label-key>=<label-value>

为k8s-node01节点打标签
kubectl label nodes k8s-node01 mariadb=mariadb

查看node节点label值
kubectl get nodes --show-labels
```

修改Label的值

语法: 需要加上`--overwrite`参数

```
kubectl label nodes <node-name> <label-key>=<label-value> --overwrite

修改k8s-node01节点label值
kubectl label nodes k8s-node01 mariadb=mariadb10.5 --overwrite

查看node节点label值
kubectl get nodes --show-labels
```

删除label语法

```
注意事项：label的可以后边要增加"-"
kubectl label nodes <node-name> <label-key>-

删除k8s-node01节点mariadb的label
kubectl label nodes k8s-node01 mariadb-

查看node节点label值
kubectl get nodes --show-labels
```

mariaDB部署

通过指定node节点label，将mariaDB部署到指定节点。方便演示volume的各种方式。

指定node

```
kubectl label nodes k8s-node01 mariadb=mariadb

查看node节点label值
kubectl get nodes --show-labels
```

service部署

```
在spec.template.spec属性下增加nodeSelector属性。
spec:
      nodeSelector: #根据label设置，配置节点选择器
        mariadb: mariadb  #语法规则: key: value
      containers:
```

全部资源文件清单

labels/mariadbsecret.yml

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: mariadbsecret
type: Opaque
data:
  password: YWRtaW4=
```

labels/mariadb.yml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mariadb-deploy
  labels:
    app: mariadb-deploy
spec:
  replicas: 1
  template:
    metadata:
      name: mariadb-deploy
      labels:
        app: mariadb-deploy
    spec:
      nodeSelector:
        mariadb: mariadb
      imagePullSecrets:
        - name: lagouharbor
      containers:
        - name: mariadb-deploy
          image: 192.168.198.155:5000/lagouedu/mariadb:10.5.2
          imagePullPolicy: IfNotPresent
          ports:
            - containerPort: 3307
```

```
          env:
            - name: MYSQL_ROOT_PASSWORD
              #这是mysqlroot用户的密码
              valueFrom:
                secretKeyRef:
                  key: password
                  name: mariadbsecret
            - name: TZ
              value: Asia/Shanghai
          args:
            - "--character-set-server=utf8mb4"
            - "--collation-server=utf8mb4_unicode_ci"
          volumeMounts:
            - mountPath: /etc/mysql/mariadb.conf.d/   #容器内的挂载目录
              name: lagoumariadb #随便给一个名字,这个名字必须与volumes.name一致
      restartPolicy: Always
      volumes:
        - name: lagoumariadb
          configMap:
            name: mariadbconfigmap
  selector:
    matchLabels:
      app: mariadb-deploy
---
apiVersion: v1
kind: Service
metadata:
  name: mariadb-svc
spec:
  selector:
    app: mariadb-deploy
  ports:
    - port: 3307
      targetPort: 3307
      nodePort: 30036
  type: NodePort
```

labels/mariadbconfigmap.yml

```
apiVersion: v1
data:
  my.cnf: "省略中间数据部分，请各位同学前面章节"
kind: ConfigMap
metadata:
  name: mariadbconfigmap


```

客户端测试

```
IP:192.168.198.157
username:root
password:admin
prot: 30036
```

volume

hostPath

hostPath类型的存储卷是指将工作节点上某文件系统的目录或文件挂载于Pod中的一种存储卷。把宿主机上的目录挂载到容器，但是在每个节点上都要有，因为不确定容器会分配到哪个节点。也是把存储从宿主机挂载到k8s集群上，但它有许多限制，例如只支持单节点（Node），而且只支持“ReadWriteOnce”模式。

指定node节点

```
kubectl label nodes k8s-node01 mariadb=mariadb

查看node节点label值
kubectl get nodes --show-labels
```

挂载卷

```
语法：
1. volumeMounts为containers下级key，containers.volumeMounts。volumes与containers平级。
2. containers.volumeMounts.name与volumes.name值一致。
3. containers.volumeMounts.mountPath是容器内目录
4. volumes.hostPath.path是宿主机挂载目录
5. volumes.hostPath.type值必须为"Directory"


containers
  volumeMounts:
    - mountPath: /var/lib/mysql
      name: mariadb-volume
  .......
volumes:
   - name: mariadb-volume
     hostPath:
        path: /data/mariadb
        type: Directory

例如：
volumeMounts:
    - mountPath: /var/lib/mysql
      name: mariadb-volume
restartPolicy: Always
```

```
volumes:
  - name: mariadb-volume
    hostPath:
      path: /data/mariadb
      type: Directory
```

全部资源文件清单

labels/mariadbsecret.yml

```
apiVersion: v1
kind: Secret
metadata:
  name: mariadbsecret
type: Opaque
data:
  password: YWRtaW4=
```

labels/mariadb.yml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mariadb-deploy
  labels:
    app: mariadb-deploy
spec:
  replicas: 1
  template:
    metadata:
      name: mariadb-deploy
      labels:
        app: mariadb-deploy
    spec:
      nodeSelector:
        mariadb: mariadb
      imagePullSecrets:
        - name: lagouharbor
      containers:
        - name: mariadb-deploy
          image: 192.168.198.155:5000/lagouedu/mariadb:10.5.2
          imagePullPolicy: IfNotPresent
          ports:
            - containerPort: 3307
          env:
```

```
            - name: MYSQL_ROOT_PASSWORD
              #这是mysqlroot用户的密码
              valueFrom:
                secretKeyRef:
                  key: password
                  name: mariadbsecret
            - name: TZ
              value: Asia/Shanghai
          args:
            - "--character-set-server=utf8mb4"
            - "--collation-server=utf8mb4_unicode_ci"
          volumeMounts:
            - mountPath: /etc/mysql/mariadb.conf.d/   #容器内的挂载目录
              name: lagoumariadb #随便给一个名字,这个名字必须与volumes.name一致
            - mountPath: /var/lib/mysql #容器内的挂载目录
              name: volume-mariadb
      restartPolicy: Always
      volumes:
        - name: lagoumariadb
          configMap:
            name: mariadbconfigmap
        - name: volume-mariadb
          hostPath:
            path: /data/mariadb
            type: Directory

  selector:
    matchLabels:
      app: mariadb-deploy
---
apiVersion: v1
kind: Service
metadata:
  name: mariadb-svc
spec:
  selector:
    app: mariadb-deploy
  ports:
    - port: 3307
      targetPort: 3307
      nodePort: 30036
  type: NodePort
```

labels/mariadbconfigmap.yml

```
apiVersion: v1
data:
  my.cnf: "省略中间数据部分，请各位同学前面章节"
kind: ConfigMap
metadata:
  name: mariadbconfigmap
```

客户端测试

```
IP:192.168.198.157
username:root
password:admin
prot: 30036
```

emptyDir

emptyDir存储卷是Pod生命周期中的一个临时目录，在pod对象被移除时会被一并删除，用得很少，例如同一pod内的多个容器间文件共享，或者作为容器数据的临时存储目录用于数据缓存系统等。同学们可以自行查找资料进行学习。

PV&&PVC

简介

部署mysql之前我们需要先了解一个概念有状态服务。这是一种特殊的服务，简单的归纳下就是会产生需要持久化的数据，并且有很强的I/O需求，且重启需要依赖上次存储到磁盘的数据。如典型的mysql，kafka，zookeeper等等。

在我们有比较优秀的商业存储的前提下，非常推荐使用有状态服务进行部署，计算和存储分离那是相当的爽的。在实际生产中如果没有这种存储，localPV也是不错的选择，当然local pv其实和hostPath是一样的。当然我们在开发测试环境也是可以自己搭建一套简单的如NFS服务，来享受存储和计算分离的爽快感。

kubernetes中定义一种了资源类型Stateful Service即有状态服务，有状态服务需要的持久化数据动态绑定我们可以利用存储的API PersistentVolume（PV）和PersistentVolumeClaim（PVC）来进行需要的相关数据的绑定和存储。

PV概念

persistentVolume：是由管理员设置的存储，它是集群的一部分。就像节点时集群中的资源一样，PV也是集群中的资源。PV是Volumes之类的卷插件，但具有独立于使用PV的pod的生命周期。此API对象包含存储实现的细节，即NFS、iSCSI或者特定于云供应商的存储系统

PVC概念

peresistentVolumeClaim是用户存储的请求。它与pod相似，pod消耗节点资源，PVC消耗PV资源。pod可以请求特定级别的资源(CPU和内存)。盛名可以请求特定的大小和访问模式。例如：可以以读/写一次或者 只读多次模式挂载。

PV & PVC

PV就好比是一个仓库，我们需要先购买一个仓库，即定义一个PV存储服务，例如CEPH,NFS,Local Hostpath等等。

PVC就好比租户，pv和pvc是一对一绑定的，挂载到POD中，一个pvc可以被多个pod挂载。

全部资源文件清单

pv

pvandpvchostpath/mariadbpv.yml

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  labels:
    app: mariadb-pv
  name: data-mariadb-pv
spec:
  accessModes:
    - ReadWriteOnce
  capacity:
    storage: 10Gi
  hostPath:
    path: /data/mariadb
    type: DirectoryOrCreate
  persistentVolumeReclaimPolicy: Retain
  storageClassName: standard
  volumeMode: Filesystem
```

pvc

pvandpvchostpath/mariadbpvc.yml

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: k8sdemo-mariadb-pvclaim
  labels:
    app: k8sdemo-mariadb-pvc
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: standard
  resources:
    requests:
      storage: 1Gi
```

service

```
    volumes:
        - name: mysqlvolume
          persistentVolumeClaim:
            claimName: k8sdemo-mariadb-pvclaim
```

完整文件信息

pvandpvchostpath/mariadb.yml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mariadb-deploy
  labels:
    app: mariadb-deploy
spec:
  replicas: 1
  template:
    metadata:
      name: mariadb-deploy
      labels:
        app: mariadb-deploy
    spec:
      nodeSelector:
        mariadb: mariadb
      imagePullSecrets:
        - name: lagouharbor
      containers:
        - name: mariadb-deploy
          image: 192.168.198.155:5000/lagouedu/mariadb:10.5.2
          imagePullPolicy: IfNotPresent
          ports:
            - containerPort: 3307
          env:
            - name: MYSQL_ROOT_PASSWORD
              #这是mysqlroot用户的密码
              valueFrom:
                secretKeyRef:
                  key: password
                  name: mariadbsecret
            - name: TZ
              value: Asia/Shanghai
          args:
            - "--character-set-server=utf8mb4"
            - "--collation-server=utf8mb4_unicode_ci"
          volumeMounts:
            - mountPath: /etc/mysql/mariadb.conf.d/   #容器内的挂载目录
              name: lagoumariadb #随便给一个名字,这个名字必须与volumes.name一致
            - mountPath: /var/lib/mysql #容器内的挂载目录
              name: volume-mariadb
      restartPolicy: Always
      volumes:
        - name: lagoumariadb
          configMap:
            name: mariadbconfigmap
```

```
        - name: volume-mariadb
          persistentVolumeClaim:
            claimName: mariadb-pvc

  selector:
    matchLabels:
      app: mariadb-deploy
---
apiVersion: v1
kind: Service
metadata:
  name: mariadb-svc
spec:
  selector:
    app: mariadb-deploy
  ports:
    - port: 3307
      targetPort: 3307
      nodePort: 30036
  type: NodePort
```

secret

pvandpvchostpath/mariadbsecret.yml

```
apiVersion: v1
kind: Secret
metadata:
  name: mariadbsecret
type: Opaque
data:
  password: YWRtaW4=
```

configmap

pvandpvchostpath/mariadb.yml

```
apiVersion: v1
data:
  my.cnf: "省略中间数据部分，请各位同学前面章节"
kind: ConfigMap
metadata:
  name: mariadbconfigmap
```

客户端测试

```
IP:192.168.198.157
username:root
password:admin
prot: 30036
```

PV&&PVC理论补充

存储机制介绍

在 Kubernetes 中，存储资源和计算资源(CPU、Memory)同样重要，Kubernetes 为了能让管理员方便管理集群中的存储资源，同时也为了让使用者使用存储更加方便，所以屏蔽了底层存储的实现细节，将存储抽象出两个 API 资源 `PersistentVolume` 和 `PersistentVolumeClaim` 对象来对存储进行管理。

- **PersistentVolume（持久化卷）：** `PersistentVolume` 简称 `PV`， 是对底层共享存储的一种抽象，将共享存储定义为一种资源，它属于集群级别资源，不属于任何 `Namespace`，用户使用 PV 需要通过 PVC 申请。PV 是由管理员进行创建和配置，它和具体的底层的共享存储技术的实现方式有关，比如说 Ceph、GlusterFS、NFS 等，都是通过插件机制完成与共享存储的对接，且根据不同的存储 PV 可配置参数也是不相同。
- **PersistentVolumeClaim（持久化卷声明）：** `PersistentVolumeClaim` 简称 `PVC`，是用户存储的一种声明，类似于对存储资源的申请，它属于一个 `Namespace` 中的资源，可用于向 `PV` 申请存储资源。`PVC` 和 `Pod` 比较类似，`Pod` 消耗的是 `Node` 节点资源，而 `PVC` 消耗的是 `PV` 存储资源，`Pod` 可以请求 CPU 和 Memory，而 `PVC` 可以请求特定的存储空间和访问模式。

上面两种资源 `PV` 和 `PVC` 的存在很好的解决了存储管理的问题，不过这些存储每次都需要管理员手动创建和管理，如果一个集群中有很多应用，并且每个应用都要挂载很多存储，那么就需要创建很多 `PV` 和 `PVC` 与应用关联。为了解决这个问题 Kubernetes 在 1.4 版本中引入了 `StorageClass` 对象。

当我们创建 `PVC` 时指定对应的 `StorageClass` 就能和 `PV` 的 `StorageClass` 关联，`StorageClass` 会交由与他关联 Provisioner 存储插件来创建与管理存储，它能帮你创建对应的 `PV` 和在远程存储上创建对应的文件夹，并且还能根据设定的参数，删除与保留数据。所以管理员只要在 `StorageClass` 中配置好对应的参数就能方便的管理集群中的存储资源。

PV 支持存储的类型

PersistentVolume 类型实现为插件,目前 Kubernetes 支持以下插件：

```
RBD：Ceph 块存储。
FC：光纤存储设备。
NFS：网络问卷存储卷。
iSCSI：iSCSI 存储设备。
CephFS：开源共享存储系统。
Flocker：一种开源共享存储系统。
Glusterfs：一种开源共享存储系统。
Flexvolume：一种插件式的存储机制。
HostPath：宿主机目录，仅能用于单机。
AzureFile：Azure 公有云提供的 File。
AzureDisk：Azure 公有云提供的 Disk。
ScaleIO Volumes：DellEMC 的存储设备。
StorageOS：StorageOS 提供的存储服务。
VsphereVolume：VMWare 提供的存储系统。
Quobyte Volumes：Quobyte 提供的存储服务。
Portworx Volumes：Portworx 提供的存储服务。
GCEPersistentDisk：GCE 公有云提供的 PersistentDisk。
AWSElasticBlockStore：AWS 公有云提供的 ElasticBlockStore。
```

PV 的生命周期

PV 生命周期总共四个阶段：

- Available（可用）—— 可用状态，尚未被 PVC 绑定。
- Bound（已绑定）—— 绑定状态，已经与某个 PVC 绑定。
- Released（已释放）—— 与之绑定的 PVC 已经被删除，但资源尚未被集群回收。
- Failed（失败）—— 当删除 PVC 清理资源，自动回收卷时失败，所以处于故障状态。
- 命令行会显示绑定到 PV 的 PVC 的名称 ——kubectl get pv命令

PV 的常用配置参数

存储能力 （capacity）

PV 可以通过配置 `capacity` 中的 `storage` 参数，对 PV 挂多大存储空间进行设置。 目前 capacity 只有一个设置存储大小的选项，未来可能会增加。

存储卷模式（volumeMode）

PV 可以通过配置 `volumeMode` 参数，对存储卷类型进行设置，可选项包括：

- **Filesystem：** 文件系统，默认是此选项。
- **Block：** 块设备

目前 Block 模式只有 AWSElasticBlockStore、AzureDisk、FC、GCEPersistentDisk、iSCSI、LocalVolume、RBD、VsphereVolume 等支持）。

访问模式（accessModes）

PV 可以通过配置 `accessModes` 参数，设置访问模式来限制应用对资源的访问权限，有以下机制访问模式：

- ReadWriteOnce——该卷可以被单个节点以读/写模式挂载。
- ReadOnlyMany——该卷可以被多个节点以只读模式挂载
- ReadWriteMany——该卷可以被多个节点以读/写模式挂载

PersistentVolume 可以以资源提供者支持的任何方式挂载到主机上。供应商具有不同的功能，每个PV 的访问模式都将被设置为该卷支持的特定模式。例如，NFS 可以支持多个读/写客户端，但特定的 NFS PV 可能以只读方式导出到服务器上。每个 PV 都有一套自己的用来描述特定功能的访问模式

-

在命令行中，访问模式缩写为：

- RWO - ReadWriteOnce
- ROX - ReadOnlyMany
- RWX - ReadWriteMany

不过不同的存储所支持的访问模式也不相同，具体如下：

Volume Plugin	ReadWriteOnce	ReadOnlyMany	ReadWriteMany
AWSElasticBlockStore	√	-	-
AzureFile	√	√	√
AzureDisk	√	-	-
CephFS	√	√	√
Cinder	√	-	-
FC	√	√	-
FlexVolume	√	√	-
Flocker	√	-	-
GCEPersistentDisk	√	√	-
GlusteFS	√	√	√
HostPath	√	-	-
iSCSI	√	√	-
PhotonPersistentDisk	√	-	-
Quobyte	√	√	√
NFS	√	√	√
RBD	√	√	-
VsphereVolume	√	-	-
PortworxVolume	√	-	√
ScaleIO	√	√	-
StorageOS	√	-	-

挂载参数（mountOptions）

PV 可以根据不同的存储卷类型，设置不同的挂载参数，每种类型的存储卷可配置参数都不相同。如 NFS 存储，可以设置 NFS 挂载配置，如下：

```
下面例子只是 NFS 支持的部分参数，其它参数请自行查找 NFS 挂载参数。
mountOptions:
  - hard
  - nfsvers=4
```

存储类 （storageClassName）

PV 可以通过配置 `storageClassName` 参数指定一个存储类 `StorageClass` 资源，具有特定 `StorageClass` 的 `PV` 只能与指定相同 `StorageClass` 的 `PVC` 进行绑定，没有设置 `StorageClass` 的 `PV` 也是同样只能与没有指定 `StorageClass` 的 `PVC` 绑定。

回收策略（persistentVolumeReclaimPolicy）

PV 可以通过配置 `persistentVolumeReclaimPolicy` 参数设置回收策略，可选项如下：

- **Retain（保留）：** 保留数据，需要由管理员手动清理。
- **Recycle（回收）：** 删除数据，即删除目录下的所有文件，比如说执行 `rm -rf /thevolume/*` 命令，目前只有 NFS 和 HostPath 支持。
- **Delete（删除）：** 删除存储资源，仅仅部分云存储系统支持，比如删除 AWS EBS 卷，目前只有 AWS EBS，GCE PD，Azure 磁盘和 Cinder 卷支持删除。

PVC常用参数

筛选器（selector）

PVC 可以通过在 `Selecter` 中设置 `Laberl` 标签，筛选出带有指定 `Label` 的 `PV` 进行绑定。`Selecter` 中可以指定 `matchLabels` 或 `matchExpressions`，如果两个字段都设定了就需要同时满足才能匹配。

```yaml
selector:
  matchLabels:
    release: "stable"
  matchExpressions:
    - key: environment
      operator: In
      values: dev
```

资源请求（resources）

PVC 设置目前只有 `requests.storage` 一个参数，用于指定申请存储空间的大小。

```yaml
resources:
  requests:
    storage: 8Gi
```

存储类（storageClass）

PVC 要想绑定带有特定 `StorageClass` 的 `PV` 时，也必须设定 `storageClassName` 参数，且名称也必须要和 `PV` 中的 `storageClassName` 保持一致。如果要绑定的 `PV` 没有设置 `storageClassName` 则 `PVC` 中也不需要设置。

当 PVC 中如果未指定 `storageClassName` 参数或者指定为空值，则还需要考虑 `Kubernetes` 中是否设置了默认的 `StorageClass`：

- 未启用 DefaultStorageClass：等于 storageClassName 值为空。
- 启用 DefaultStorageClass：等于 storageClassName 值为默认的 StorageClass。
- 如果设置 storageClassName=""，则表示该 PVC 不指定 StorageClass。

访问模式（accessModes）

PVC 中可设置的访问模式与 PV 种一样，用于限制应用对资源的访问权限。

NFS存储卷

NFS介绍

NFS 是 Network FileSystem 的缩写，顾名思义就是网络文件存储系统， 分为服务端（Server）和客户端（Client）。最早由 sun 公司开发，是类 unix 系统间实现磁盘共享的一种方法。 它允许网络中的计算机之间通过 TCP/IP 网络共享资源。通过 NFS，我们本地 NFS 的客户端应用可以透明地读写位于服务端 NFS 服务器上的文件，就像访问本地文件一样方便。简单的理解，NFS 就是可以透过网络，让不同的主机、不同的操作系统可以共享存储的服务。

NFS 在文件传送或信息传送过程中依赖于 RPC（Remote Procedure Call） 协议，即远程过程调用，NFS 的各项功能都必须要向 RPC 来注册，如此一来 RPC 才能了解 NFS 这个服务的各项功能 Port、PID、NFS 在服务器所监听的 IP 等，而客户端才能够透过 RPC 的询问找到正确对应的端口，所以，NFS 必须要有 RPC 存在时才能成功的提供服务，简单的理解二者关系：NFS是 一个文件存储系统，而 RPC 是负责信息的传输。

NFS共享存储方式

- 手动方式静态创建所需要的PV和PVC。
- 通过创建PVC动态地创建对应PV，无需手动创建PV。

NFS安装

k8s集群所有节点都需要安装NFS服务。本章节实验我们选用k8s的master节点作为NFS服务的server端。

```
yum install -y nfs-utils rpcbind
```

创建共享目录

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

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


vi /etc/exports
/nfs/mariadb *(rw,no_root_squash,no_all_squash,sync)
```

参数说明

参数	说明
`ro`	只读访问
`rw`	读写访问
`sync`	所有数据在请求时写入共享
`async`	nfs 在写入数据前可以响应请求
`secure`	nfs 通过 1024 以下的安全 TCP/IP 端口发送
`insecure`	nfs 通过 1024 以上的端口发送
`wdelay`	如果多个用户要写入 nfs 目录，则归组写入（默认）
`no_wdelay`	如果多个用户要写入 nfs 目录，则立即写入，当使用 async 时，无需此设置
`hide`	在 nfs 共享目录中不共享其子目录
`no_hide`	共享 nfs 目录的子目录
`subtree_check`	如果共享 /usr/bin 之类的子目录时，强制 nfs 检查父目录的权限（默认）
`no_subtree_check`	不检查父目录权限
`all_squash`	共享文件的 UID 和 GID 映射匿名用户 anonymous，适合公用目录
`no_all_squash`	保留共享文件的 UID 和 GID（默认）
`root_squash`	root 用户的所有请求映射成如 anonymous 用户一样的权限（默认）
`no_root_squash`	root 用户具有根目录的完全管理访问权限
`anonuid=xxx`	指定 nfs 服务器 /etc/passwd 文件中匿名用户的 UID
`anongid=xxx`	指定 nfs 服务器 /etc/passwd 文件中匿名用户的 GID

启动NFS服务

k8s集群所有节点启动NFS服务。

```
systemctl start rpcbind
systemctl start nfs

设置开启启动
systemctl enable rpcbind
systemctl enable nfs
```

测试NFS服务

```
在另一台 Linux 虚拟机上测试一下，是否能够正确挂载:
showmount -e 192.168.198.156

在客户端创建挂在目录
mkdir -p /data/mariadb

挂载远端目录到本地 /data/mariadb 目录
mount 192.168.198.156:/nfs/mariadb /data/mariadb


NFS服务端写入
$ echo "This is NFS server." > /nfs/mariadb/nfs.txt

客户端读取
cat /data/mariadb/nfs.txt


客户端写入
$ echo "This is NFS client." >> /data/mariadb/nfs.txt

服务端读取
$ cat /nfs/mariadb/nfs.txt

都是没问题的，这是因为上边设置了 NFS 远端目录权限为 rw 拥有读写权限，如果设置为 ro，那么客户端只能读取，不能写入，根据实际应用场景合理配置，这里就不在演示了。这里提一下，NFS 默认使用 UDP 协议来进行挂载，为了提高 NFS 的稳定性，可以使用 TCP 协议挂载，那么客户端挂载命令可使用如下命令

mount 192.168.198.156:/nfs/mysql /data/mysql -o proto=tcp -o nolock
```

客户端卸载 NFS 挂载目录

```
umount /data/mariadb/

强制卸载
umount  -l /data/mariadb/
```

NFS4服务

```
使用NFS4协议方式进行多共享目录配置。所有共享目录的根目录为/nfs/data。服务器端的/etc/exports文件中的配置为:
vi /etc/exports

/nfs/data *(rw,fsid=0,sync,no_wdelay,insecure_locks,no_root_squash)

K8S的静态NFS服务PV的nfs:path 的值不用写共享根目录，直接写/mariadb即可。K8S会帮我们配置成/nfs/data/mariadb目录


重启NFS
systemctl restart rpcbind
systemctl restart nfs
```

pv配置

```
  mountOptions:
    - hard
    - nfsvers=4.1
  nfs:
    path: /mariadb
    server: 192.168.198.156
```

全部配置文件清单

pv

nfs/mariadbpv.yml

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: data-mariadb-pv
  labels:
    app: mariadb-pv
spec:
  accessModes:
    - ReadWriteOnce
  capacity:
    storage: 10Gi
  mountOptions:
```

```
    - hard
    - nfsvers=4.1
  nfs:
    path: /mariadb
    server: 192.168.198.156
  persistentVolumeReclaimPolicy: Retain
  storageClassName: standard
  volumeMode: Filesystem
```

pvc

nfs/mariadbpvc.yml

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mariadb-pvc
  labels:
    app: mariadb-pvc
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: standard
  resources:
    requests:
      storage: 5Gi
```

service

nfs/mariadb.yml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mariadb-deploy
  labels:
    app: mariadb-deploy
spec:
  replicas: 1
  template:
    metadata:
      name: mariadb-deploy
      labels:
        app: mariadb-deploy
    spec:
      imagePullSecrets:
        - name: lagouharbor
      containers:
        - name: mariadb-deploy
          image: 192.168.198.155:5000/lagouedu/mariadb:10.5.2
          imagePullPolicy: IfNotPresent
          ports:
            - containerPort: 3307
```

```
          env:
            - name: MYSQL_ROOT_PASSWORD
              #这是mysqlroot用户的密码
              valueFrom:
                secretKeyRef:
                  key: password
                  name: mariadbsecret
            - name: TZ
              value: Asia/Shanghai
          args:
            - "--character-set-server=utf8mb4"
            - "--collation-server=utf8mb4_unicode_ci"
          volumeMounts:
            - mountPath: /etc/mysql/mariadb.conf.d/   #容器内的挂载目录
              name: lagoumariadb #随便给一个名字,这个名字必须与volumes.name一致
            - mountPath: /var/lib/mysql #容器内的挂载目录
              name: volume-mariadb
      restartPolicy: Always
      volumes:
        - name: lagoumariadb
          configMap:
            name: mariadbconfigmap
        - name: volume-mariadb
          persistentVolumeClaim:
            claimName: mariadb-pvc

  selector:
    matchLabels:
      app: mariadb-deploy
---
apiVersion: v1
kind: Service
metadata:
  name: mariadb-svc
spec:
  selector:
    app: mariadb-deploy
  ports:
    - port: 3307
      targetPort: 3307
      nodePort: 30036
  type: NodePort
```

secret

nfs/mariadbsecret.yml

```
apiVersion: v1
kind: Secret
metadata:
  name: mariadbsecret
type: Opaque
data:
  password: YWRtaW4=
```

configmap

nfs/mariadb.yml

```
apiVersion: v1
data:
  my.cnf: "省略中间数据部分，请各位同学前面章节"
kind: ConfigMap
metadata:
  name: mariadbconfigmap
```

客户端测试

```
IP:192.168.198.157
username:root
password:admin
prot: 30036
```

集群调度

k8s内pod由scheduler调度，scheduler的任务是把pod分配到合适的node节点上。scheduler调度时会考虑到node节点的资源使用情况、port使用情况、volume使用情况等等...在此基础之上，我们也可以控制pod的调度。

Scheduler 是 kubernetes 调度器，主要的任务是把定义的 pod 分配到集群的节点上。但要很多要考虑的问题：

- 公平：如何保证每个节点都能被合理分配资源，不要造成一个节点忙死，一个节点闲死局面。
- 资源高效利用：集群所有资源最大化被使用。内存、硬盘、CPU等因素。
- 效率：调度的性能要好，能够尽快地对大批量的 pod 完成调度工作。
- 灵活：允许用户根据自己的需求控制调度的逻辑

> Sheduler 是作为单独的程序运行，启动之后会一直与 API Server保持通讯，获取 PodSpec.NodeName 为空的 pod，对每个 pod 都会创建一个 binding，表明该 pod 应该放到哪个节点上 。

固定节点

Pod.spec.nodeSelector

```
前边的课程已经给大家介绍过

关键技能点：
1.给某一个节点打标签
kubectl label nodes k8s-node01 mariadb=mariadb


```

```
2.pod的控制器中增加配置属性

...
    spec:
      nodeSelector:
        mariadb: mariadb
...
```

Pod.spec.nodeName

```
删除k8s-node01节点mariadb的label
kubectl label nodes k8s-node02 mariadb-

kubectl label nodes k8s-node02 --show-labels
```

修改label案例用于演示nodeName属性

pod控制器关键代码

```
    spec:
      nodeName: k8s-node02
```

全部资源文件清单

controller

labels/mariadb.yml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mariadb-deploy
  labels:
    app: mariadb-deploy
spec:
  replicas: 1
  template:
    metadata:
      name: mariadb-deploy
      labels:
        app: mariadb-deploy
    spec:
      nodeName: k8s-node02
      imagePullSecrets:
        - name: lagouharbor
      containers:
        - name: mariadb-deploy
          image: 192.168.198.155:5000/lagouedu/mariadb:10.5.2
          imagePullPolicy: IfNotPresent
          ports:
            - containerPort: 3307
```

```
          env:
            - name: MYSQL_ROOT_PASSWORD
              #这是mysqlroot用户的密码
              valueFrom:
                secretKeyRef:
                  key: password
                  name: mariadbsecret
            - name: TZ
              value: Asia/Shanghai
          args:
            - "--character-set-server=utf8mb4"
            - "--collation-server=utf8mb4_unicode_ci"
          volumeMounts:
            - mountPath: /etc/mysql/mariadb.conf.d/   #容器内的挂载目录
              name: lagoumariadb #随便给一个名字,这个名字必须与volumes.name一致
      restartPolicy: Always
      volumes:
        - name: lagoumariadb
          configMap:
            name: mariadbconfigmap
  selector:
    matchLabels:
      app: mariadb-deploy
---
apiVersion: v1
kind: Service
metadata:
  name: mariadb-svc
spec:
  selector:
    app: mariadb-deploy
  ports:
    - port: 3307
      targetPort: 3307
      nodePort: 30036
  type: NodePort
```

secret

labels/mariadbsecret.yml

```
apiVersion: v1
kind: Secret
metadata:
  name: mariadbsecret
type: Opaque
data:
  password: YWRtaW4=
```

configmap

labels/mariadb.yml

```yaml
apiVersion: v1
data:
  my.cnf: "省略中间数据部分，请各位同学前面章节"
kind: ConfigMap
metadata:
  name: mariadbconfigmap
```

集群调度原理

Scheduler调度步骤

1. 首先用户在通过 Kubernetes 客户端 Kubectl 提交创建 Pod 的 Yaml 的文件，向Kubernetes 系统发起资源请求，该资源请求被提交到Kubernetes 系统。
2. Kubernetes 系统中，用户通过命令行工具 Kubectl 向 Kubernetes 集群即 APIServer 用 的方式发送“POST”请求，即创建 Pod 的请求。
3. APIServer 接收到请求后把创建 Pod 的信息存储到 Etcd 中，从集群运行那一刻起，资源调度系统 Scheduler 就会定时去监控 APIServer
4. 通过 APIServer 得到创建 Pod 的信息，Scheduler 采用 watch 机制，一旦 Etcd 存储 Pod 信息成功便会立即通知APIServer，
5. APIServer会立即把Pod创建的消息通知Scheduler，Scheduler发现 Pod 的属性中 Dest Node 为空时（Dest Node=""）便会立即触发调度流程进行调度。
6. **而这一个创建Pod对象，在调度的过程当中有3个阶段：节点预选、节点优选、节点选定，从而筛选出最佳的节点**
    - 节点预选：基于一系列的预选规则对每个节点进行检查，将那些不符合条件的节点过滤，从而完成节点的预选
    - 节点优选：对预选出的节点进行优先级排序，以便选出最合适运行Pod对象的节点
    - 节点选定：从优先级排序结果中挑选出优先级最高的节点运行Pod，当这类节点多于1个时，则进行随机选择

集群调度策略

Kubernetes调度器作为集群的大脑，在如何提高集群的资源利用率、保证集群中服务的稳定运行中也会变得越来越重要Kubernetes的资源分为两种属性。

1. 可压缩资源（例如CPU循环，Disk I/O带宽）都是可以被限制和被回收的，对于一个Pod来说可以降低这些资源的使用量而不去杀掉Pod。
2. 不可压缩资源（例如内存、硬盘空间）一般来说不杀掉Pod就没法回收。未来Kubernetes会加入更多资源，如网络带宽，存储IOPS的支持。

常用预选策略

预选策略	作用
CheckNodeCondition	检查是否可以在节点报告磁盘、网络不可用或未准备好时将Pod调度其上
HostName	如果Pod对象拥有spec.hostname属性，则检查节点名称字符串是否和该属性值匹配。
PodFitsHostPorts	Pod的spec.hostPort属性时，检查端口是否被占用
MatchNodeSelector	Pod的spec.nodeSelector属性时，检查节点标签
NoDiskConflict	Pod依赖的存储卷在此节点是否可用，默认没有启用
PodFitsResources	检查节点上的资源（CPU、内存）可用性是否满足Pod对象的运行需求。
PodToleratesNodeTaints	Pod的spec.tolerations属性，仅关注NoSchedule和NoExecute两个效用标识的污点
PodToleratesNodeNoExecuteTaints	Pod的spec.tolerations属性，是否能接纳节点的NoExecute类型污点,默认没有启用
CheckNodeLabelPresence	仅检查节点上指定的所有标签的存在性,默认没有启用
CheckServiceAffinity	将相同Service的Pod对象放置在同一个或同一类节点上以提高效率,默认没有启用
MaxEBSVolumeCount	检查节点已挂载的EBS(亚马逊弹性块存储)存储卷数量是否超过设置的最大值，默认为39
MaxGCEPDVolumeCount	检查节点上已挂载的GCE PD（谷歌云存储） 存储卷数量是否超过最大值，默认为16
MaxAzureDiskVolumeCount	检查节点上已挂载的Azure Disk存储卷数量是否超过最大值，默认为16
CheckVolumeBinding	检查节点上已绑定和未绑定的PVC是否满足需求
NoVolumeZoneConflict	在给定区域zone限制下，检查此节点部署的Pod对象是否存在存储卷冲突
CheckNodeMemoryPressure	检查节点内存压力，如果压力过大，那就不会讲pod调度至此
CheckPodePIDPressure	检查节点PID资源压力
CheckNodeDiskPressure	检查节点磁盘资源压力
MatchInterPodAffinity	检查节点是否满足Pod对象亲和性或反亲和性条件

常用优先函数

函数名称	详细说明
LeastRequestedPriority	节点的优先级就由节点空闲资源与节点总容量的比值，即由（总容量-节点上Pod的容量总和-新Pod的容量）/总容量）来决定。 CPU和内存具有相同权重，资源空闲比越高的节点得分越高。 `cpu((capacity – sum(requested)) * 10 / capacity) + memory((capacity – sum(requested)) * 10 / capacity) / 2`
BalancedResourceAllocation	CPU和内存使用率越接近的节点权重越高，该策略不能单独使用，必须和LeastRequestedPriority组合使用，尽量选择在部署Pod后各项资源更均衡的机器。 如果请求的资源（CPU或者内存）需求大于节点的capacity，那么该节点永远不会被调度到。
InterPodAffinityPriority	通过迭代 weightedPodAffinityTerm 的元素计算和，并且如果对该节点满足相应的PodAffinityTerm，则将 “weight” 加到和中，具有最高和的节点是最优选的。
SelectorSpreadPriority	为了更好的容灾，对同属于一个service、replication controller或者replica的多个Pod副本，尽量调度到多个不同的节点上。 如果指定了区域，调度器则会尽量把Pod分散在不同区域的不同节点上。当一个Pod的被调度时，会先查找Pod对于的service或者replication controller， 然后查找service或replication controller中已存在的Pod，运行Pod越少的节点的得分越高。本质就是往运行同类pod少的节点上分配。
NodeAffinityPriority	亲和性机制。Node Selectors（调度时将pod限定在指定节点上）， 支持多种操作符（In, NotIn, Exists, DoesNotExist, Gt, Lt），而不限于对节点labels的精确匹配。 另外支持两种类型的选择器，一种是“hard（requiredDuringSchedulingIgnoredDuringExecution）”选择器，它保证所选的主机必须满足所有Pod对主机的规则要求。 这种选择器更像是之前的nodeselector，在nodeselector的基础上增加了更合适的表现语法。另一种是“soft（preferresDuringSchedulingIgnoredDuringExecution）”选择器， 它作为对调度器的提示，调度器会尽量但不保证满足NodeSelector的所有要求。
NodePreferAvoidPodsPriority（权重1W）	如果 节点的 Anotation （注解信息）没有设置 key-value:scheduler.alpha.kubernetes.io/ preferAvoidPods = "…"，则节点对该 policy 的得分就是10分， 加上权重10000，那么该node对该policy的得分至少10W分。如果Node的Anotation设置了，scheduler.alpha.kubernetes.io/preferAvoidPods = "…" ，如果该 pod 对应的 Controller 是 ReplicationController 或 ReplicaSet， 则该 node 对该 policy 的得分就是0分。
TaintTolerationPriority	使用 Pod 中 tolerationList 与 节点 Taint 列表项进行匹配，配对成功的项越多，则得分越低。污点越匹配，得分越低
ImageLocalityPriority	根据Node上是否存在一个pod的容器运行所需镜像大小对优先级打分，分值为0-10。遍历全部Node， 如果某个Node上pod容器所需的镜像一个都不存在，分值为0； 如果Node上存在Pod容器部分所需镜像，则根据满足当前需求的镜像的大小来决定分值，镜像越大，分值就越高；如果Node上存在pod所需全部镜像，分值为10。默认没有启用
EqualPriority	是一个优先级函数，它给予所有节点相等权重。
MostRequestedPriority	在 ClusterAutoscalerProvider 中，替换 LeastRequestedPriority，给使用多资源的节点，更高的优先级。 计算公式为： `(cpu(10 sum(requested) / capacity) + memory(10 sum(requested) / capacity)) / 2`默认没有启用

节点亲和性调度

节点亲和性规则：

required(硬亲和性，不能商量，必须执行) 、preferred(软亲和性，可以商量，选择执行)。

- 硬亲和性规则不满足时，Pod会置于Pending状态，软亲和性规则不满足时，会选择一个不匹配的节点
- 当节点标签改变而不再符合此节点亲和性规则时，不会将Pod从该节点移出，仅对新建的Pod对象生效

节点硬亲和性

requiredDuringSchedulingIgnoredDuringExecution

- 方式一：Pod使用 spec.nodeSelector (基于等值关系);Pod使用 spec.nodeName
- 方式二：Pod使用 spec.affinity 支持matchExpressions属性 (复杂标签选择机制)

全部资源文件清单

controller

labels/mariadb.yml，删除spec.selectNode或者spec.nodeName信息。

技能点概述:Pod.sepc.affinity

```
    affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
              - matchExpressions:
                  - key: kubernetes.io/hostname    #node节点的标签
                    operator: In
                    values:
                      - k8s-node02    #集群真实节点名称
```

```
可以先使用命令获得节点标签及真实节点名称:
kubectl get nodes --show-labels
```

键值运算关系

```
In：label 的值在某个列表中
NotIn：label 的值不在某个列表中
Gt：label 的值大于某个值
Lt：label 的值小于某个值
Exists：某个 label 存在
DoesNotExist：某个 label 不存在
```

全部文件清单

```
apiVersion: apps/v1
kind: Deployment
```

```yaml
metadata:
  name: mariadb-deploy
  labels:
    app: mariadb-deploy
spec:
  replicas: 1
  template:
    metadata:
      name: mariadb-deploy
      labels:
        app: mariadb-deploy
    spec:
      imagePullSecrets:
        - name: lagouharbor
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
              - matchExpressions:
                  - key: kubernetes.io/hostname   #node节点的标签
                    operator: In
                    values:
                      - k8s-node02
      containers:
        - name: mariadb-deploy
          image: 192.168.198.155:5000/lagouedu/mariadb:10.5.2
          imagePullPolicy: IfNotPresent
          ports:
            - containerPort: 3307
          env:
            - name: MYSQL_ROOT_PASSWORD
              #这是mysqlroot用户的密码
              valueFrom:
                secretKeyRef:
                  key: password
                  name: mariadbsecret
            - name: TZ
              value: Asia/Shanghai
          args:
            - "--character-set-server=utf8mb4"
            - "--collation-server=utf8mb4_unicode_ci"
          volumeMounts:
            - mountPath: /etc/mysql/mariadb.conf.d/   #容器内的挂载目录
              name: lagoumariadb #随便给一个名字,这个名字必须与volumes.name一致
      restartPolicy: Always
      volumes:
        - name: lagoumariadb
          configMap:
            name: mariadbconfigmap
  selector:
    matchLabels:
      app: mariadb-deploy
---
apiVersion: v1
kind: Service
metadata:
  name: mariadb-svc
spec:
```

```
  selector:
    app: mariadb-deploy
  ports:
    - port: 3307
      targetPort: 3307
      nodePort: 30036
  type: NodePort
```

secret

labels/mariadbsecret.yml

```
apiVersion: v1
kind: Secret
metadata:
  name: mariadbsecret
type: Opaque
data:
  password: YWRtaW4=
```

configmap

labels/mariadb.yml

```
apiVersion: v1
data:
  my.cnf: "省略中间数据部分，请各位同学前面章节"
kind: ConfigMap
metadata:
  name: mariadbconfigmap
```

错误节点信息

如果我们选择的节点不存在，pod状态会一直处于Pending。例如：

Pod.sepc.affinity

```
   affinity:
       nodeAffinity:
         requiredDuringSchedulingIgnoredDuringExecution:
           nodeSelectorTerms:
             - matchExpressions:
                 - key: kubernetes.io/hostname    #node节点的标签
                   operator: In
                   values:
                     - k8s-node05    #集群真实节点名称
```

```
集群中不存在k8s-node05的节点。当我们部署服务时，查看pod信息，会发现pod一直处于Pending

kubectl apply -f .

kubectl get pods -o wide

查看pod详细信息：发现提示没有节点的提示。
kubectl describe pods mariadb-deploy-9d5457866-rxcr2
```

节点软亲和性

preferredDuringSchedulingIgnoredDuringExecution

- 柔性控制逻辑，当条件不满足时，能接受被编排于其他不符合条件的节点之上
- 权重 weight 定义优先级，1-100 值越大优先级越高

全部资源文件清单

controller

labels/mariadb.yml，删除spec.selectNode或者spec.nodeName信息。

技能点概述:Pod.sepc.affinity

```yaml
    affinity:
        nodeAffinity:
          preferredDuringSchedulingIgnoredDuringExecution:
            - preference:
                matchExpressions:
                  - key: kubernetes.io/hostname
                    operator: In
                    values:
                      - k8s-node02
              weight: 1
```

```
可以先使用命令获得节点标签及真实节点名称:
kubectl get nodes --show-labels
```

键值运算关系

```
In：label 的值在某个列表中
NotIn：label 的值不在某个列表中
Gt：label 的值大于某个值
Lt：label 的值小于某个值
Exists：某个 label 存在
DoesNotExist：某个 label 不存在
```

全部文件清单

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mariadb-deploy
  labels:
    app: mariadb-deploy
spec:
  replicas: 1
  template:
    metadata:
      name: mariadb-deploy
      labels:
        app: mariadb-deploy
    spec:
      nodeSelector:
        mariadb: mariadb
      imagePullSecrets:
        - name: lagouharbor
      affinity:
        nodeAffinity:
          preferredDuringSchedulingIgnoredDuringExecution:
            - preference:
                matchExpressions:
                  - key: kubernetes.io/hostname
                    operator: In
                    values:
                      - k8s-node02
              weight: 1
      containers:
        - name: mariadb-deploy
          image: 192.168.198.155:5000/lagouedu/mariadb:10.5.2
          imagePullPolicy: IfNotPresent
          ports:
            - containerPort: 3307
          env:
            - name: MYSQL_ROOT_PASSWORD
              #这是mysqlroot用户的密码
              valueFrom:
                secretKeyRef:
                  key: password
                  name: mariadbsecret
            - name: TZ
              value: Asia/Shanghai
          args:
            - "--character-set-server=utf8mb4"
            - "--collation-server=utf8mb4_unicode_ci"
          volumeMounts:
            - mountPath: /etc/mysql/mariadb.conf.d/   #容器内的挂载目录
              name: lagoumariadb #随便给一个名字,这个名字必须与volumes.name一致
      restartPolicy: Always
      volumes:
        - name: lagoumariadb
```

```
          configMap:
            name: mariadbconfigmap
  selector:
    matchLabels:
      app: mariadb-deploy
---
apiVersion: v1
kind: Service
metadata:
  name: mariadb-svc
spec:
  selector:
    app: mariadb-deploy
  ports:
    - port: 3307
      targetPort: 3307
      nodePort: 30036
  type: NodePort
```

secret

labels/mariadbsecret.yml

```
apiVersion: v1
kind: Secret
metadata:
  name: mariadbsecret
type: Opaque
data:
  password: YWRtaW4=
```

configmap

labels/mariadb.yml

```
apiVersion: v1
data:
  my.cnf: "省略中间数据部分，请各位同学前面章节"
kind: ConfigMap
metadata:
  name: mariadbconfigmap
```

错误节点信息

如果我们选择的节点不存在，pod状态会一直处于Pending。例如：

Pod.sepc.affinity

```yaml
    affinity:
        nodeAffinity:
          preferredDuringSchedulingIgnoredDuringExecution:
            - preference:
                matchExpressions:
                  - key: kubernetes.io/hostname
                    operator: In
                    values:
                      - k8s-node05
              weight: 1
```

```
集群中不存在k8s-node05的节点。当我们部署服务时，查看pod信息，会发现pod一直处于Pending

kubectl apply -f .

kubectl get pods -o wide

查看pod详细信息：发现提示没有节点的提示。
kubectl describe pods mariadb-deploy-9d5457866-rxcr2
```

Pod资源亲和调度

Pod硬亲和调度

requiredDuringSchedulingIgnoredDuringExecution

Pod亲和性描述一个Pod与具有某特征的现存Pod运行位置的依赖关系；即需要事先存在被依赖的Pod对象

Pod软亲和调度

Pod软亲和调度用于分散同一类应用，调度至不同的区域、机架或节点等.将 `spec.affinity.podAffinity` 替换为 `spec.affinity.podAntiAffinity`。软亲和调度也分为柔性约束和强制约束

污点和容忍度

污点 `taints 是定义在node节点` 上的键值型属性数据，用于让节点拒绝将Pod调度运行于其上，除非Pod有接纳节点污点的容忍度。容忍度 `tolerations 是定义在Pod` 上的键值属性数据，用于配置可容忍的污点，且调度器将Pod调度至其能容忍该节点污点的节点上或没有污点的节点上 。

对于nodeAffinity无论是硬策略(硬亲和)还是软策略(软亲和)方式，都是调度 pod 到预期节点上，而 Taints恰好与之相反，如果一个节点标记为 Taints ，除非 pod 也被标识为可以容忍污点节点，否则该 Taints 节点不会被调度 pod。

节点亲和性，是 pod 的一种属性（偏好或硬性要求），它使 pod 被吸引到一类特定的节点。Taint 则相反，它使
节点 能够 排斥 一类特定的 pod
Taint 和 toleration 相互配合，可以用来避免 pod 被分配到不合适的节点上。每个节点上都可以应用一个或多个
taint ，这表示对于那些不能容忍这些 taint 的 pod，是不会被该节点接受的。如果将 toleration 应用于 pod
上，则表示这些 pod 可以（但不要求）被调度到具有匹配 taint 的节点上

定义污点和容忍度

污点定义于 `nodes.spec.taints` 属性。容忍度定义于 `pods.spec.tolerations` 属性。

使用 kubectl taint 命令可以给某个 Node 节点设置污点，Node 被设置上污点之后就和 Pod 之间存在了一种相
斥的关系，可以让 Node 拒绝 Pod 的调度执行，甚至将 Node 已经存在的 Pod 驱逐出去 。

语法： `key=value:effect`

```
查看node节点名称
kubectl get nodes


查看master节点详细信息：通过观taints察属性，发现master节点默认被打上一个污点。
kubectl describe nodes k8s-master01
```

effect定义排斥等级：

- `NoSchedule`，不能容忍，但仅影响调度过程，已调度上去的pod不受影响，仅对新增加的pod生效。

  解释说明：表示 k8s 将不会将 Pod 调度到具有该污点的 Node 上 。
- `PreferNoSchedule`，柔性约束，节点现存Pod不受影响，如果实在是没有符合的节点，也可以调度上来。 解释说明：表示 k8s 将不会将 Pod 调度到具有该污点的 Node 上 。
- `NoExecute`，不能容忍，当污点变动时，Pod对象会被驱逐。

  解释说明：表示 k8s 将不会将 Pod 调度到具有该污点的 Node 上，同时会将 Node 上已经存在的 Pod 驱逐出去

全部资源文件清档

本案例用于演示创建、删除污点及驱逐pod的过程。

污点语法

```
创建污点:语法规则
kubectl taint nodes node1 key1=value1:NoSchedule

删除污点:语法规则
kubectl taint nodes node1 key1:NoSchedule-
```

deploymentdemo控制器

产生10个副本

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: deploymentdemo
  labels:
    app: deploymentdemo
spec:
  replicas: 10
  template:
    metadata:
      name: deploymentdemo
      labels:
        app: deploymentdemo
    spec:
      containers:
        - name: deploymentdemo
          image: nginx:1.17.10-alpine
          imagePullPolicy: IfNotPresent
          ports:
            - containerPort: 80
      restartPolicy: Always
  selector:
    matchLabels:
      app: deploymentdemo
```

设置污点

```
观察每个节点pod运行情况
kubectl get pods -o wide

在某一个节点创建污点并驱逐pod
kubectl taint nodes k8s-node03 offline=testtaint:NoExecute


查看pod被驱逐过程
kubectl get pods -o wide

删除污点
kubectl taint nodes k8s-node03 offline=testtaint:NoExecute-
```

```
查看节点污点信息
kubectl describe nodes k8s-node03
```

在Pod上定义容忍度时：

1. 等值比较 容忍度与污点在key、value、effect三者完全匹配
2. 存在性判断 key、effect完全匹配，value使用空值

一个节点可配置多个污点，一个Pod也可有多个容忍度

全部资源文件清档

本案例用于演示创建、删除及驱逐pod的过程。

######

设置污点

```
在某一个节点创建污点
kubectl taint nodes k8s-node03 offline=testtaint:NoSchedule

查看节点污点信息
kubectl describe nodes k8s-node03
```

deploymentdemo控制器

产生10个副本

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: deploymentdemo
  labels:
    app: deploymentdemo
spec:
  replicas: 10
  template:
    metadata:
      name: deploymentdemo
      labels:
        app: deploymentdemo
    spec:
      containers:
```

```
        - name: deploymentdemo
          image: nginx:1.17.10-alpine
          imagePullPolicy: IfNotPresent
          ports:
            - containerPort: 80
      restartPolicy: Always
  selector:
    matchLabels:
      app: deploymentdemo
```

查看部署情况

```
部署控制器
kubectl apply -f deploymentdemo.yml

查看是否有pod被部署到k8s-node03节点
kubectl get pods -o wide

删除控制器
kubectl delete -f deploymentdemo.yml
```

设置pod容忍度

```
Pod.spec.tolerations属性:
...
    spec:
      tolerations:
        - key: "offline"
          operator: "Equal"
          value: "testtaint"
          effect: "NoSchedule"
      containers:
...
```

完整控制器清单

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: deploymentdemo
  labels:
    app: deploymentdemo
spec:
  replicas: 10
  template:
    metadata:
      name: deploymentdemo
      labels:
        app: deploymentdemo
    spec:
      tolerations:
        - key: "offline"
```

```
          operator: "Equal"
          value: "testtaint"
          effect: "NoSchedule"
      containers:
        - name: deploymentdemo
          image: nginx:1.17.10-alpine
          imagePullPolicy: IfNotPresent
          ports:
            - containerPort: 80
      restartPolicy: Always
  selector:
    matchLabels:
      app: deploymentdemo
```

部署控制器

```
部署控制器
kubectl apply -f deploymentdemo.yml

查看pod详细信息
kubectl get pods -o wide

删除控制器
kubectl delete -f deploymentdemo.yml


删除污点
kubectl taint nodes k8s-node03 offline=testtaint:NoSchedule-
```