**原创；微信公众号：千里行走；**

**受限图片大小限制，有些图片不是很清晰，可以到微信公众号查看；**

实际生产环境中，为了稳定和高可用(晚上睡觉踏实)，我们并不会把mysql装在k8s集群中，一般是用阿里云的RDS或者自己在高性能机器上搭建mysql。

但是，对于dev, test环境，我们完全可以把mysql部署到各自的k8s集群,非常有助于提升效率，而且还有助于k8s的经验积累.

**目录**

**(1).helm方式部署**

**(2).组件解析**

**(3).yaml方式部署**

**(4).yaml文件详解**

1.mysql-min-storageclass-local.yaml

2.mysql-min-pv-local.yaml

3.mysql-min-pvc.yaml

4.mysql-min-secret.yaml

5.mysql-min-deployment.yaml

6.mysql-min-service.yaml

7.mysql-min-pv-nfs.yaml

**(5).参考**

**正文**

**(1).helm方式部署**

由于mysql要持久化数据，所以是有状态容器，我们要先对pv做实例：

kubectl apply -f mysql-min-pv-local.yaml

kubectl apply -f mysql-min-storageclass-local.yaml

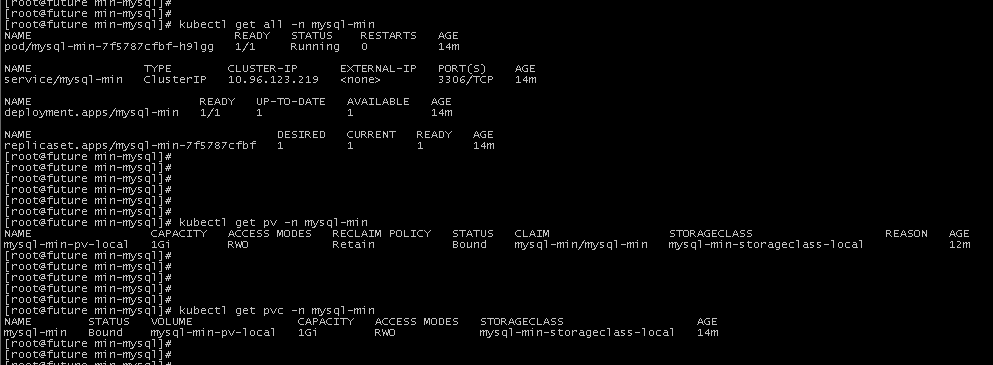
pv与storageClass的yaml文件参见：

<https://github.com/hepyu/k8s-app-config/tree/master/helm/min-cluster-allinone/min-mysql>

然后执行helm：

helm install --name mysql-min stable/mysql --namespace=mysql-min --set "persistence.size=1Gi,persistence.storageClass=mysql-min-storageclass-local"

执行kubectl get all -n mysql-min等相关命令查看容器中的mysql：



容器外部访问k8s中的mysql，通过pod-forward：

kubectl port-forward svc/mysql-min 3306 -n mysql-min --address 0.0.0.0

获得root用户的密码：

kubectl get secret --namespace mysql-min mysql-min -o jsonpath="{.data.mysql-root-password}" | base64 --decode; echo

然后执行登录命令登入k8s内部的mysql：

mysql -h 127.0.0.1 -uroot -p9ZeNk0DghH

**(2).组件解析**

从1中的图中可以看到相关组件有：pod, service, deployment.apps, replicaset.apps, pv, pvc。

Pod：运行真实mysql实例的docker容器；

Service：容器内部的负载均衡，service指向pod；比如如果我们在容器中部署apollo配置中心，那么apollo的jdbc的域名部分就是这个service的名字，port就是service中的port。

Deployment.apps：定义了mysql在容器中的规格，比如cpu, memory等资源占用，部署几个容器，以及部署方式(滚动部署还是all stop all start等方式，默认是RollingUpdate，滚动升级)；当然对于一个mysql实例来说不存在多容器，只有一个，也就无所谓RollingUpdate。

**(3).yaml方式部署**

笔者提供了一组可正常部署的yaml文件，git地址：

https://github.com/hepyu/k8s-app-config/tree/master/yaml/min-cluster-allinone/mysql-min

顺次执行(或者参照readme.MD执行)：

kubectl apply -f mysql-min-storageclass-local.yaml

kubectl apply -f mysql-min-pv-local.yaml

kubectl apply -f mysql-min-pvc.yaml

kubectl apply -f mysql-min-secret.yaml

kubectl apply -f mysql-min-deployment.yaml

kubectl apply -f mysql-min-service.yaml

**(4).yaml文件详解**

也可以笔者的git，查看对应的yaml文件(都有注释)：

<https://github.com/hepyu/k8s-app-config/tree/master/yaml/min-cluster-allinone/mysql-min>

1.mysql-min-storageclass-local.yaml

kind: StorageClass

apiVersion: storage.k8s.io/v1

metadata:

name: mysql-min-storageclass-local

#指定存储类的供应者,比如aws, nfs等，具体取值参考官方说明。

#存储类有一个供应者的参数域，此参数域决定PV使用什么存储卷插件。参数必需进行设置

#由于demo中使用的是本地存储，所以这里写kubernetes.io/no-provisioner.

provisioner: kubernetes.io/no-provisioner

#volumeBindingMode 参数将延迟PVC绑定，直到 pod 被调度。

volumeBindingMode: WaitForFirstConsumer

2.mysql-min-pv-local.yaml

apiVersion: v1

kind: PersistentVolume

metadata:

name: mysql-min-pv-local

namespace: mysql

spec:

capacity:

storage: 1Gi

volumeMode: Filesystem

accessModes:

- ReadWriteOnce

storageClassName: "mysql-min-storageclass-local"

persistentVolumeReclaimPolicy: Retain

#表示使用本地存储

local:

path: /datavip/k8s-data/mysql-min-pv-local

#使用local pv时必须定义nodeAffinity，Kubernetes Scheduler需要使用PV的nodeAffinity描述信息来保证Pod能够调度到有对应local volume的Node上。

#创建local PV之前，你需要先保证有对应的storageClass已经创建。

nodeAffinity:

required:

nodeSelectorTerms:

- matchExpressions:

- key: kubernetes.io/hostname

operator: In

values:

#future是pod需要分不到的主机名，这台主机上开启了local-pv资源。

- future

3.mysql-min-pvc.yaml

apiVersion: v1

items:

- apiVersion: v1

kind: PersistentVolumeClaim

metadata:

#当启用PVC 保护 alpha 功能时，如果用户删除了一个 pod 正在使用的 PVC，则该 PVC 不会被立即删除。PVC 的删除将被推迟，直到 PVC 不再被任何 pod 使用。

#可以看到，当 PVC 的状态为 Teminatiing 时，PVC 受到保护，Finalizers 列表中包含 kubernetes.io/pvc-protection：

finalizers:

- kubernetes.io/pvc-protection

labels:

app: mysql-min

release: mysql-min

name: mysql-min

namespace: mysql-min

spec:

#PV 的访问模式（accessModes）有三种：

#ReadWriteOnce（RWO）：是最基本的方式，可读可写，但只支持被单个 Pod 挂载。

#ReadOnlyMany（ROX）：可以以只读的方式被多个 Pod 挂载。

#ReadWriteMany（RWX）：这种存储可以以读写的方式被多个 Pod 共享。

accessModes:

- ReadWriteOnce

resources:

requests:

storage: 1Gi

storageClassName: mysql-min-storageclass-local

#表示使用本地磁盘，实际生产中一般都使用nfs。

volumeMode: Filesystem

volumeName: mysql-min-pv-local

kind: List

4.mysql-min-secret.yaml

apiVersion: v1

data:

#将mysql数据库的所有user的password配置到secret，统一管理

mysql-password: YWRtaW4=

mysql-root-password: OVplTmswRGdoSA==

kind: Secret

metadata:

labels:

app: mysql-min

release: mysql-min

name: mysql-min

namespace: mysql-min

#Secret有三种类型：

#Opaque：base64编码格式的Secret，用来存储密码、密钥等；但数据也通过base64 –decode解码得到原始数据，所有加密性很弱。

#kubernetes.io/dockerconfigjson：用来存储私有docker registry的认证信息。

#kubernetes.io/service-account-token： 用于被serviceaccount引用。serviceaccout创建时Kubernetes会默认创建对应的secret。Pod如果使用了serviceaccount，对应的secret会自动挂载到Pod目录/run/secrets/ kubernetes.io/serviceaccount中。

type: Opaque

5.mysql-min-deployment.yaml

apiVersion: apps/v1

kind: Deployment

metadata:

annotations:

deployment.kubernetes.io/revision: "1"

generation: 1

labels:

app: mysql-min

release: mysql-min

name: mysql-min

namespace: mysql-min

spec:

replicas: 1

selector:

matchLabels:

app: mysql-min

strategy:

rollingUpdate:

maxSurge: 1 #滚动升级时会先启动1个pod

maxUnavailable: 1 #滚动升级时允许的最大Unavailable的pod个数

type: RollingUpdate #滚动升级

template:

metadata:

labels:

app: mysql-min

spec:

containers:

- env:

#password存储在secret中

- name: MYSQL\_ROOT\_PASSWORD

valueFrom:

secretKeyRef:

key: mysql-root-password

name: mysql-min

- name: MYSQL\_PASSWORD

valueFrom:

secretKeyRef:

key: mysql-password

name: mysql-min

- name: MYSQL\_USER

value: apollo

- name: MYSQL\_DATABASE

image: mysql:5.7.14

imagePullPolicy: IfNotPresent

#kubelet 使用 liveness probe（存活探针）来确定何时重启容器。例如，当应用程序处于运行状态但无法做进一步操作，liveness 探针将捕获到 deadlock，重启处于该状态下的容器，使应用程序在存在 bug 的情况下依然能够继续运行下去

livenessProbe:

exec:

command:

- sh

- -c

- mysqladmin ping -u root -p${MYSQL\_ROOT\_PASSWORD}

failureThreshold: 3 #探测成功后，最少连续探测失败多少次才被认定为失败。默认是 3。最小值是 1。

initialDelaySeconds: 30 #容器启动后第一次执行探测是需要等待多少秒。

periodSeconds: 10 #执行探测的频率。默认是10秒，最小1秒。

successThreshold: 1 #探测失败后，最少连续探测成功多少次才被认定为成功。默认是 1。对于 liveness 必须是 1。最小值是 1。

timeoutSeconds: 5 #探测超时时间。默认1秒，最小1秒。

name: mysql-min

ports:

- containerPort: 3306

name: mysql

protocol: TCP

#Kubelet 使用 readiness probe（就绪探针）来确定容器是否已经就绪可以接受流量。只有当 Pod 中的容器都处于就绪状态时 kubelet 才会认定该 Pod处于就绪状态。该信号的作用是控制哪些 Pod应该作为service的后端。如果 Pod 处于非就绪状态，那么它们将会被从 service 的 load balancer中移除。

readinessProbe:

exec:

command:

- sh

- -c

- mysqladmin ping -u root -p${MYSQL\_ROOT\_PASSWORD}

failureThreshold: 3

initialDelaySeconds: 5

periodSeconds: 10

successThreshold: 1

timeoutSeconds: 1

resources:

requests:

cpu: 100m

memory: 256Mi

#为了达到一个相当高水平的实用性，特别是为了积极开发应用，快速调试失败是很重要的。除了一般的日志采集，Kubernetes还能通过查出重大错误原因来加速调试，并在某种程度上通过kubectl或者UI陈列出来。可以指定一个’terminationMessagePath’来让容器写下它的“death rattle“，比如声明失败消息，堆栈跟踪，免责条款等等。默认途径是‘/dev/termination-log’。

terminationMessagePath: /dev/termination-log

# 此字段默认为 “File“，这意味着仅从终止消息文件中检索终止消息。 通过将 terminationMessagePolicy 设置为 “FallbackToLogsOnError“，你就可以告诉 Kubernetes，在容器因错误退出时，如果终止消息文件为空，则使用容器日志输出的最后一块作为终止消息。 日志输出限制为 2048 字节或 80 行，以较小者为准。

terminationMessagePolicy: File

#要使用的数据盘目录，在initContainer中会关联此处目录。

volumeMounts:

- mountPath: /var/lib/mysql

name: data

dnsPolicy: ClusterFirst

#Init 容器支持应用容器的全部字段和特性，包括资源限制、数据卷和安全设置。 然而，Init 容器对资源请求和限制的处理稍有不同，在下面 资源 处有说明。 而且 Init 容器不支持 Readiness Probe，因为它们必须在 Pod 就绪之前运行完成。

#如果为一个 Pod 指定了多个 Init 容器，那些容器会按顺序一次运行一个。 每个 Init 容器必须运行成功，下一个才能够运行。 当所有的 Init 容器运行完成时，Kubernetes 初始化 Pod 并像平常一样运行应用容器。

#mysql这里的initContainer是为了保证在POD启动前，PV盘要先行绑定成功。

initContainers:

- command:

- rm

- -fr

- /var/lib/mysql/lost+found

image: busybox:1.29.3

imagePullPolicy: IfNotPresent

name: remove-lost-found

resources: {}

terminationMessagePath: /dev/termination-log

terminationMessagePolicy: File

volumeMounts:

- mountPath: /var/lib/mysql

name: data

restartPolicy: Always

#scheduler 是 kubernetes 的调度器，主要的任务是把定义的 pod 分配到集群的节点上。

schedulerName: default-scheduler

securityContext: {}

#如果您的Pod通常需要超过30秒才能关闭，请确保增加优雅终止宽限期。可以通过在Pod YAML中设置terminationGracePeriodSeconds选项来实现.

#如果容器在优雅终止宽限期后仍在运行，则会发送SIGKILL信号并强制删除。与此同时，所有的Kubernetes对象也会被清除。

terminationGracePeriodSeconds: 30

#定义数据卷PVC,与PV匹配。

volumes:

- name: data

persistentVolumeClaim:

claimName: mysql-min

6.mysql-min-service.yaml

apiVersion: v1

kind: Service

metadata:

labels:

app: mysql-min

release: mysql-min

name: mysql-min

namespace: mysql-min

spec:

ports:

- name: mysql

port: 3306

protocol: TCP

targetPort: mysql

selector:

app: mysql-min

#目前sessionAffinity可以提供"None"與"ClientIP"两种设定:

#None: 以round robin的方式轮询下面的Pods。

#ClientIP: 以client ip的方式固定request到同一台机器。

sessionAffinity: None

type: ClusterIP

7.mysql-min-pv-nfs.yaml

apiVersion: v1

kind: PersistentVolume

metadata:

name: mysql-min-pv-local

namespace: mysql

spec:

capacity:

storage: 1Gi

volumeMode: Filesystem

accessModes:

- ReadWriteOnce

storageClassName: "mysql-min-storageclass-nfs"

persistentVolumeReclaimPolicy: Retain

#存储采用nfs

nfs:

path: /mysql-min

server: xxx.nas.aliyuncs.com

容器外部登录K8S内部的mysql的方式与前边helm节所述相同。

**(5).参考**

1.helm/charts官方

<https://github.com/helm/charts/tree/master/stable/mysql>

2.kubernetes官网文档