MySQL

# 安装

## 通过Docker进行安装

# 先创建文件夹

mkdir -p /data/mysql

# 创建容器

docker run \

-u root \

--name mysql \

--restart=always \

-d \

-p 33060:3306 \

-e MYSQL\_ROOT\_PASSWORD=123456 \

-v /data/mysql:/var/lib/mysql \

mysql:5.7

mysqld.cnf文件位置

/etc/mysql/mysql.conf.d/mysqld.cnf

## 通过Kubernetes进行安装

### 创建名称空间和文件夹

kubectl create ns mysql

mkdir -p /usr/local/kubernetes/mysql

### 创建 mysql-deploy.yaml

vi /usr/local/kubernetes/mysql/mysql-deploy.yaml

apiVersion: apps/v1

kind: StatefulSet

metadata:

name: mysql

namespace: mysql

spec:

selector:

matchLabels:

app: mysql

serviceName: mysql

replicas: 3

template:

metadata:

labels:

app: mysql

spec:

initContainers:

- name: init-mysql

image: mysql:5.7

command:

- bash

- "-c"

- |

set -ex

# 从Pod的序号，生成server-id

[[ `hostname` =~ -([0-9]+)$ ]] || exit 1

ordinal=${BASH\_REMATCH[1]}

echo [mysqld] > /mnt/conf.d/server-id.cnf

# 由于server-id=0有特殊含义，我们给ID加一个100来避开它

echo server-id=$((100 + $ordinal)) >> /mnt/conf.d/server-id.cnf

# 如果Pod序号是0，说明它是Master节点，从ConfigMap里把Master的配置文件拷贝到/mnt/conf.d/目录；

# 否则，拷贝Slave的配置文件

if [[ $ordinal -eq 0 ]]; then

cp /mnt/config-map/master.cnf /mnt/conf.d/

else

cp /mnt/config-map/slave.cnf /mnt/conf.d/

fi

volumeMounts:

- name: conf

mountPath: /mnt/conf.d

- name: config-map

mountPath: /mnt/config-map

- name: clone-mysql

image: registry.cn-hangzhou.aliyuncs.com/rehighness/kubernetes:xtrabackup

command:

- bash

- "-c"

- |

set -ex

# 拷贝操作只需要在第一次启动时进行，所以如果数据已经存在，跳过

[[ -d /var/lib/mysql/mysql ]] && exit 0

# Master节点(序号为0)不需要做这个操作

[[ `hostname` =~ -([0-9]+)$ ]] || exit 1

ordinal=${BASH\_REMATCH[1]}

[[ $ordinal -eq 0 ]] && exit 0

# 使用ncat指令，远程地从前一个节点拷贝数据到本地

ncat --recv-only mysql-$(($ordinal-1)).mysql 3307 | xbstream -x -C /var/lib/mysql

# 执行--prepare，这样拷贝来的数据就可以用作恢复了

xtrabackup --prepare --target-dir=/var/lib/mysql

volumeMounts:

- name: data

mountPath: /var/lib/mysql

subPath: mysql

- name: conf

mountPath: /etc/mysql/conf.d

- name: mysql-conf

mountPath: /etc/mysql/mysql.conf.d

containers:

- name: mysql

image: mysql:5.7

resources:

requests:

memory: 200Mi

env:

- name: MYSQL\_ALLOW\_EMPTY\_PASSWORD

value: "1"

ports:

- name: mysql

containerPort: 3306

volumeMounts:

- name: data

mountPath: /var/lib/mysql

subPath: mysql

- name: conf

mountPath: /etc/mysql/conf.d

- name: mysql-conf

mountPath: /etc/mysql/mysql.conf.d

livenessProbe:

exec:

command: ["mysqladmin", "ping"]

initialDelaySeconds: 30

periodSeconds: 10

timeoutSeconds: 5

readinessProbe:

exec:

# 通过TCP连接的方式进行健康检查

command: ["mysql", "-h", "127.0.0.1", "-e", "SELECT 1"]

initialDelaySeconds: 5

periodSeconds: 2

timeoutSeconds: 1

- name: xtrabackup

image: registry.cn-hangzhou.aliyuncs.com/rehighness/kubernetes:xtrabackup

ports:

- name: xtrabackup

containerPort: 3307

command:

- bash

- "-c"

- |

set -ex

cd /var/lib/mysql

# 从备份信息文件里读取MASTER\_LOG\_FILEM和MASTER\_LOG\_POS这两个字段的值，用来拼装集群初始化SQL

if [[ -f xtrabackup\_slave\_info ]]; then

# 如果xtrabackup\_slave\_info文件存在，说明这个备份数据来自于另一个Slave节点。这种情况下，XtraBackup工具在备份的时候，就已经在这个文件里自动生成了"CHANGE MASTER TO" SQL语句。所以，我们只需要把这个文件重命名为change\_master\_to.sql.in，后面直接使用即可

mv xtrabackup\_slave\_info change\_master\_to.sql.in

# 所以，也就用不着xtrabackup\_binlog\_info了

rm -f xtrabackup\_binlog\_info

elif [[ -f xtrabackup\_binlog\_info ]]; then

# 如果只存在xtrabackup\_binlog\_inf文件，那说明备份来自于Master节点，我们就需要解析这个备份信息文件，读取所需的两个字段的值

[[ `cat xtrabackup\_binlog\_info` =~ ^(.\*?)[[:space:]]+(.\*?)$ ]] || exit 1

rm xtrabackup\_binlog\_info

# 把两个字段的值拼装成SQL，写入change\_master\_to.sql.in文件

echo "CHANGE MASTER TO MASTER\_LOG\_FILE='${BASH\_REMATCH[1]}',\

MASTER\_LOG\_POS=${BASH\_REMATCH[2]}" > change\_master\_to.sql.in

fi

# 如果change\_master\_to.sql.in，就意味着需要做集群初始化工作

if [[ -f change\_master\_to.sql.in ]]; then

# 但一定要先等MySQL容器启动之后才能进行下一步连接MySQL的操作

echo "Waiting for mysqld to be ready (accepting connections)"

until mysql -h 127.0.0.1 -e "SELECT 1"; do sleep 1; done

echo "Initializing replication from clone position"

# 将文件change\_master\_to.sql.in改个名字，防止这个Container重启的时候，因为又找到了change\_master\_to.sql.in，从而重复执行一遍这个初始化流程

mv change\_master\_to.sql.in change\_master\_to.sql.orig

# 使用change\_master\_to.sql.orig的内容，也是就是前面拼装的SQL，组成一个完整的初始化和启动Slave的SQL语句

mysql -h 127.0.0.1 <<EOF

$(<change\_master\_to.sql.orig),

MASTER\_HOST='mysql-0.mysql',

MASTER\_USER='root',

MASTER\_PASSWORD='',

MASTER\_CONNECT\_RETRY=10;

START SLAVE;

EOF

fi

# 使用ncat监听3307端口。它的作用是，在收到传输请求的时候，直接执行"xtrabackup --backup"命令，备份MySQL的数据并发送给请求者

exec ncat --listen --keep-open --send-only --max-conns=1 3307 -c \

"xtrabackup --backup --slave-info --stream=xbstream --host=127.0.0.1 --user=root"

volumeMounts:

- name: data

mountPath: /var/lib/mysql

subPath: mysql

- name: conf

mountPath: /etc/mysql/conf.d

- name: mysql-conf

mountPath: /etc/mysql/mysql.conf.d

volumes:

- name: conf

emptyDir: {}

- name: mysql-conf

configMap:

name: mysql-conf

- name: config-map

configMap:

name: mysql

volumeClaimTemplates:

- metadata:

name: data

annotations:

volume.beta.kubernetes.io/storage-class: course-nfs-storage

spec:

accessModes: [ "ReadWriteOnce" ]

resources:

requests:

storage: 100Gi

### 创建 mysql-service.yaml

cat > /usr/local/kubernetes/mysql/mysql-service.yaml << EOF

apiVersion: v1

kind: Service

metadata:

name: mysql

namespace: mysql

labels:

app: mysql

spec:

ports:

- name: mysql

port: 3306

clusterIP: None

selector:

app: mysql

---

apiVersion: v1

kind: Service

metadata:

name: mysql-read

namespace: mysql

labels:

app: mysql

spec:

type: ClusterIP

ports:

- name: mysql

port: 3306

selector:

app: mysql

EOF

### 创建 mysql-conf.yaml

cat > /usr/local/kubernetes/mysql/mysql-conf.yaml << EOF

apiVersion: v1

kind: ConfigMap

metadata:

name: mysql

namespace: mysql

labels:

app: mysql

data:

master.cnf: |

# 主节点MySQL的配置文件

[mysqld]

log-bin

slave.cnf: |

# 从节点MySQL的配置文件，该配置实现Slave节点拒绝除数据同步以外的所有写操作

[mysqld]

super-read-only

---

apiVersion: v1

kind: ConfigMap

metadata:

name: mysql-conf

namespace: mysql

labels:

app: mysql

data:

mysqld.cnf: |

# 配置utf8

[mysqld]

pid-file = /var/run/mysqld/mysqld.pid

socket = /var/run/mysqld/mysqld.sock

datadir = /var/lib/mysql

symbolic-links=0EOF

[mysqld]

character-set-server=utf8

collation-server=utf8\_general\_ci

lower\_case\_table\_names=1

EOF

### 创建 mysql-proxy.yaml（暴露无头服务）

cat > /usr/local/kubernetes/mysql/mysql-proxy.yaml << EOF

apiVersion: v1

kind: ConfigMap

metadata:

name: proxy

namespace: mysql

labels:

app: proxy

data:

nginx.conf : |

user nginx;

worker\_processes 1;

error\_log /var/log/nginx/error.log warn;

pid /var/run/nginx.pid;

events {

worker\_connections 1024;

}

http {

include /etc/nginx/mime.types;

default\_type application/octet-stream;

log\_format main '$remote\_addr - $remote\_user [$time\_local] "$request" '

'$status $body\_bytes\_sent "$http\_referer" '

'"$http\_user\_agent" "$http\_x\_forwarded\_for"';

access\_log /var/log/nginx/access.log main;

sendfile on;

#tcp\_nopush on;

keepalive\_timeout 65;

#gzip on;

include /etc/nginx/conf.d/\*.conf;

}

stream {

upstream cloudsocket {

hash $remote\_addr consistent;

server mysql-0.mysql:3306 weight=5 max\_fails=3 fail\_timeout=30s;

}

server {

listen 3306;#数据库服务器监听端口

proxy\_connect\_timeout 10s;

proxy\_timeout 300s;#设置客户端和代理服务之间的超时时间，如果5分钟内没操作将自动断开。

proxy\_pass cloudsocket;

}

}

---

apiVersion: apps/v1

kind: Deployment

metadata:

labels:

app: nginx

name: nginx

namespace: mysql

spec:

progressDeadlineSeconds: 600

replicas: 1

revisionHistoryLimit: 10

selector:

matchLabels:

app: nginx

template:

metadata:

labels:

app: nginx

spec:

containers:

- image: nginx

imagePullPolicy: Always

name: nginx

ports:

- containerPort: 80

protocol: TCP

resources:

requests:

memory: 200Mi

terminationMessagePath: /dev/termination-log

terminationMessagePolicy: File

volumeMounts:

- mountPath: /etc/nginx/nginx.conf

name: nginx

subPath: nginx.conf

restartPolicy: Always

securityContext:

seLinuxOptions: {}

terminationGracePeriodSeconds: 30

volumes:

- configMap:

defaultMode: 420

items:

- key: nginx.conf

path: nginx.conf

name: proxy

name: nginx

---

apiVersion: v1

kind: Service

metadata:

annotations:

k8s.kuboard.cn/workload: nginx

labels:

app: nginx

name: nginx

namespace: mysql

spec:

clusterIP: 10.96.79.56

ports:

- name: nginx

nodePort: 3306

port: 3306

protocol: TCP

targetPort: 3306

selector:

app: nginx

sessionAffinity: None

type: NodePort

EOF

### 创建MySQL集群

kubectl create ns mysql \

&& kubectl apply -f /usr/local/kubernetes/mysql/mysql-conf.yaml \

&& kubectl apply -f /usr/local/kubernetes/mysql/mysql-proxy.yaml \

&& kubectl apply -f /usr/local/kubernetes/mysql/mysql-service.yaml \

&& kubectl apply -f /usr/local/kubernetes/mysql/mysql-deploy.yaml

### 测试MySQL集群

尝试向这个 MySQL 集群发起请求，执行一些 SQL 操作来验证它是否正常。

在跨越名称空间进行访问时，需要在后面附带要访问的名称空间。例如：mysql-read.mysql

kubectl run mysql-client --image=mysql:5.7 -i --rm --restart=Never --\

mysql -h mysql-0.mysql.mysql <<EOF

CREATE DATABASE test;

CREATE TABLE test.messages (message VARCHAR(250));

INSERT INTO test.messages VALUES ('hello');

EOF

通过启动一个容器，使用 MySQL client 执行了创建数据库和表、以及插入数据的操作。需要注意的是，连接的 MySQL 的地址必须是 mysql-0.mysql（即：Master 节点的 DNS 记录）。因为，只有 Master 节点才能处理写操作。

而通过连接 mysql-read 这个 Service就可以用 SQL 进行读操作。

kubectl run mysql-client --image=mysql:5.7 -i -t --rm --restart=Never --\

mysql -h mysql-read.mysql -e "SELECT \* FROM test.messages"

Waiting for pod default/mysql-client to be running, status is Pending, pod ready: false

+---------+

| message |

+---------+

| hello |

+---------+

pod "mysql-client" deleted

StatefulSet也可以像 Deployment 那样非常方便地扩展这个 MySQL 集群。

kubectl scale statefulset mysql --replicas=5

## 初始化

### Docker安装初始化配置

# 修改配置文件

# 确认配置文件是644权限（chmod 644 mysqld.cnf）

docker cp mysql:/etc/mysql/mysql.conf.d/mysqld.cnf mysqld.cnf

cat >> mysqld.cnf << EOF

[mysqld]

character-set-server=utf8

collation-server=utf8\_general\_ci

lower\_case\_table\_names=1

EOF

docker cp mysqld.cnf mysql:/etc/mysql/mysql.conf.d/mysqld.cnf

docker exec -it mysql bash

# 登录mysql

mysql -u root -p

# 查看数据库字符集

SHOW VARIABLES LIKE 'character\_set\_%';

# 修改字符集

SET NAMES 'utf8';

## 报错状态强制启动

在mysqld.cnf最后一行添加

innodb\_force\_recovery = 1

如果不行，则换成2再次尝试，直到6为止。

Innodb\_force\_recovery 可以设置6个非零值：

1. (SRV\_FORCE\_IGNORE\_CORRUPT):忽略检查到的corrupt页。
2. (SRV\_FORCE\_NO\_BACKGROUND):阻止主线程的运行，如主线程需要执行full purge操作，会导致crash。
3. (SRV\_FORCE\_NO\_TRX\_UNDO):不执行事务回滚操作。
4. (SRV\_FORCE\_NO\_IBUF\_MERGE):不执行插入缓冲的合并操作。
5. (SRV\_FORCE\_NO\_UNDO\_LOG\_SCAN):不查看重做日志，InnoDB存储引擎会将未提交的事务视为已提交。
6. (SRV\_FORCE\_NO\_LOG\_REDO):不执行前滚的操作。