**OpenStack trove原理及配置实践**

2017-03-22 20:24[技术](http://www.sohu.com/tag/65790)

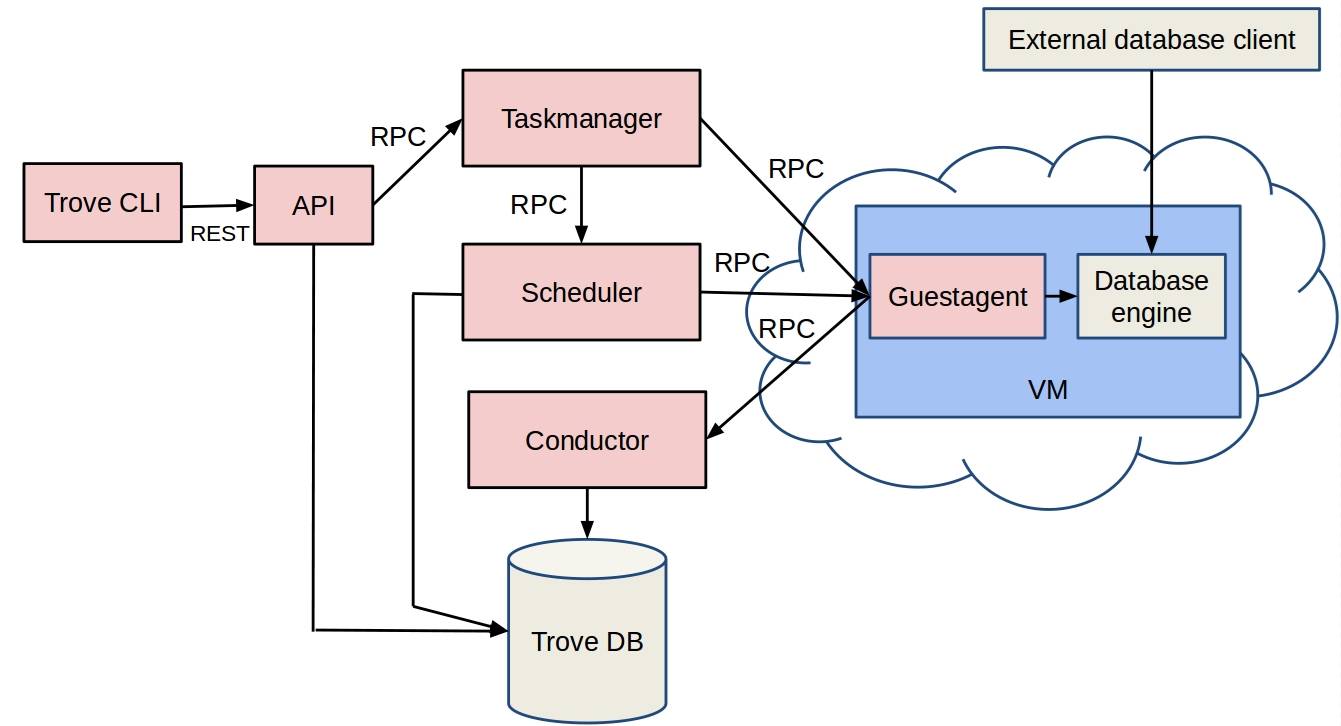
**DBaaS是什么？**

字面上理解数据库即是服务，简单来说就是以服务的形式为用户提供数据库服务。

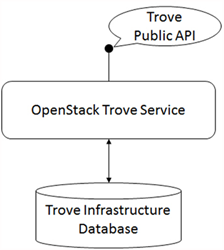
**在云平台上使用trove有什么优势？**

1. 简化IT操作流程，降低使用数据库使用门槛举个例子，曾经我搭建一个LAMP网站，数据库要自己安装，创建，授权，必要的话，还要自己做主从很繁琐，而且不是专业人员也搞不定，有了Dbaas后，我只需要在控制台点几下就弄好。
2. 自动化操作，自动的增、删、改、备。
3. 更好的资源利用，你可以根据业务量，自由的对数据库实例进行伸缩。

架构解析



trove和其他一些openstak组件一样，它暴露一个public-api，通过这个api访问trove-service，同时也保存着一些数据库实例状态到数据库中。



组件功能

* trove-api

用于操作请求的接收和分发操作提供REST风格的API，同时与trove-conductor和trove-taskmanager通信，一些轻量级的请求比如获取实例状态，实例数量等操作都是自身直接处理或访问trove

conductor和trove-taskmanager处理，比较重量级的操作比如创建数据库，创建备份等操作都是通过rpc传递给trove-taskmanager，taskmanager，然后在通过调用nova、swift、neutron、cinder等组件来完成操作。

* trove-conductor

将vm内trove-guestagent发送的状态信息保存到数据库，与trove-guestagent的通信是通过rpc来实现的，trove-conductor这个组件的目的是为了避免创建的数据库的实例直接访问数据库，它是做为一个trove-guestagent将昨天写入数据库的中间件。

* trove-taskmanager

执行trove中大部分复杂的操作，请求者发送消息到task manager，task manager在请求者的上下文中调用相应的程序执行这些请求。task manager处理一些操作，包括实例的创建、删除，与其他服务如Nova、Cinder、Swift等的交互，一些更复杂的Trove操作如复制和集群，以及对实例的整个生命周期的管理。trov-taskmanager就像是其他openstak服务的客户端，如nova，swift，cinder等，当要创建数据库实例时就将请求发送给nova，让nova去创建个实例，要备份的话就调用swift接口上传备份。

* trove-guestagent

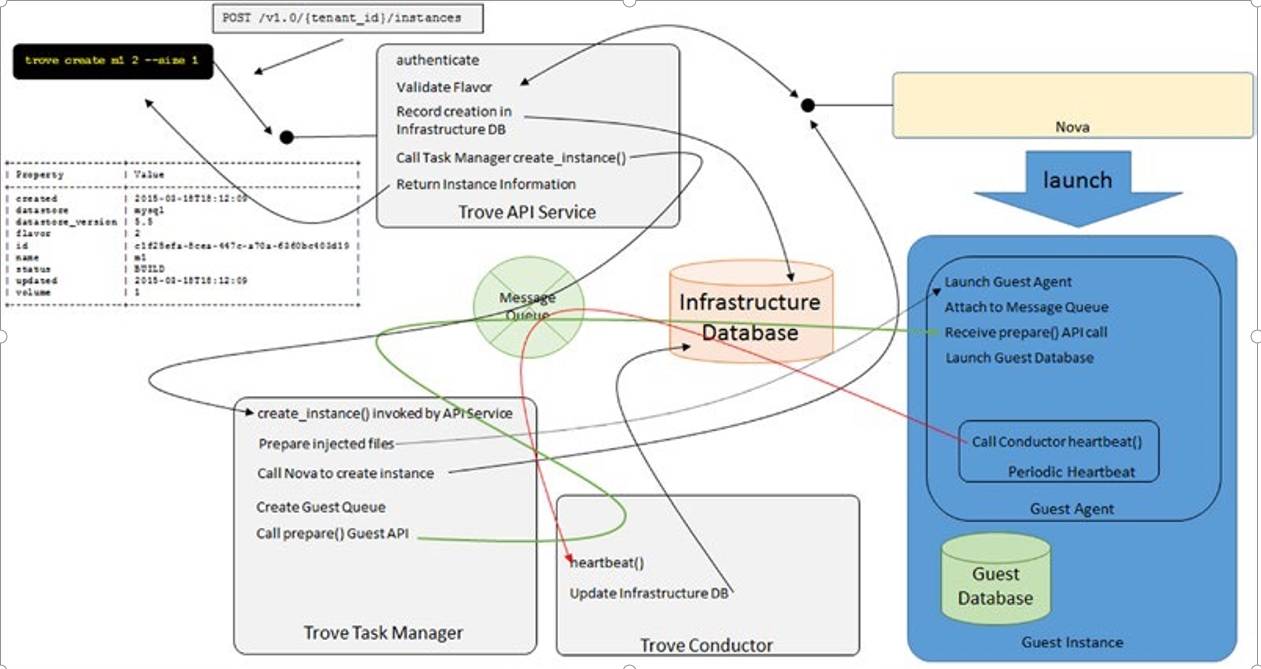
trove-guestagent集成在vm镜像里面，通过监听rpc里面task manager发过来的指令，并在本地执行代码完成数据库任务，taskmanager将消息发送到guest agent，guest agent通过调用相应的程序执行这些请求。

功能原理介绍（**这里只介绍对mysql数据库的功能实现，因为trove对mysql支持比较成熟**）

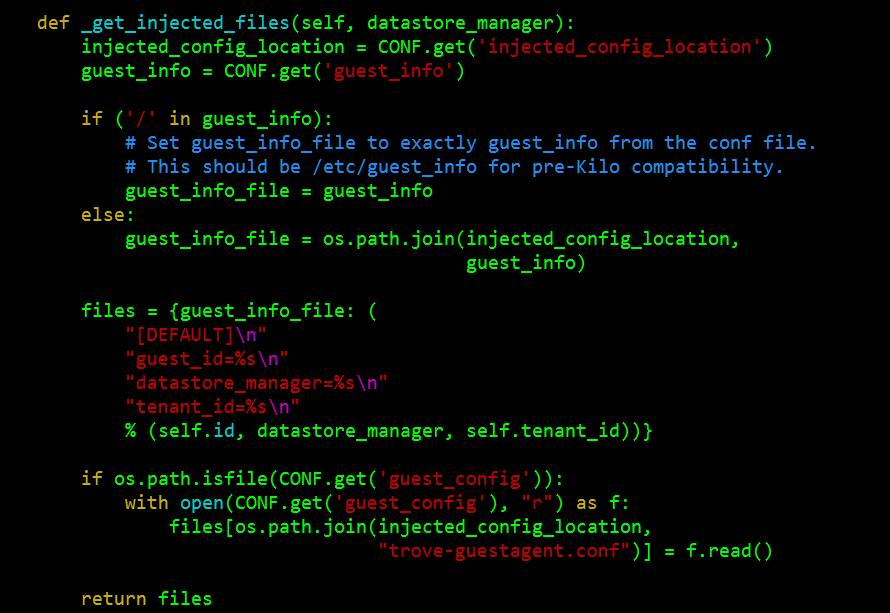
这里分别介绍三个功能的原理：

1. 创建数据库实例
2. 创建数据库备份
3. mysql的主从

**创建数据库实例**



创建数据库实例时，实际上就是通过**trove-taskmanager create\_instance()**方法去调用**nova-api**，然后调用**\_get\_injected\_files**方法将**guet\_info**和**trove-guetagent.conf**信息注入到 数据库实例**/etc/trove/conf.d/**里面，提供给**guest-agent**进行后续的操作。



所以4.0版本的trove并不需要一开始就将trove-guestagent.conf这个配置文件封装在镜像里面，这个配置文件是通过nova注入的，所以镜像只需要配置好guest-agent从哪里读这个配置文件。剩下的就交给trove-guestagent guest\_info\_file这个配置文件。

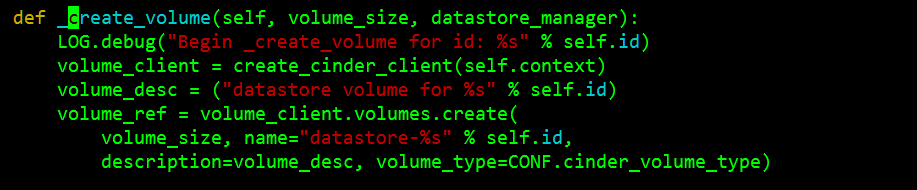
[DEFAULT]

guest\_id=7ec35639-5139-4ae4-8388-8101e41cc0f7 #这个ID是trove 分配给这个实例的ID

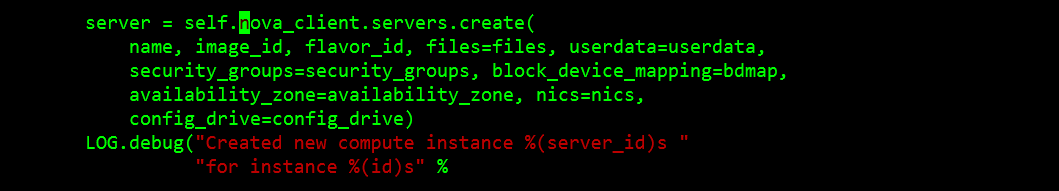
datastore\_manager=mysql #采用的是哪个datastore

tenant\_id=f2f0e038ff0342a3bc99d8971f829ac2 #是哪个租户的

当你在控制台输入需要创建的云硬盘的大小时，实际上是通过调用taskmanager里面的\_create\_volume方法：



收集齐上面那些信息后，然后调用nova来创建数据库实例：



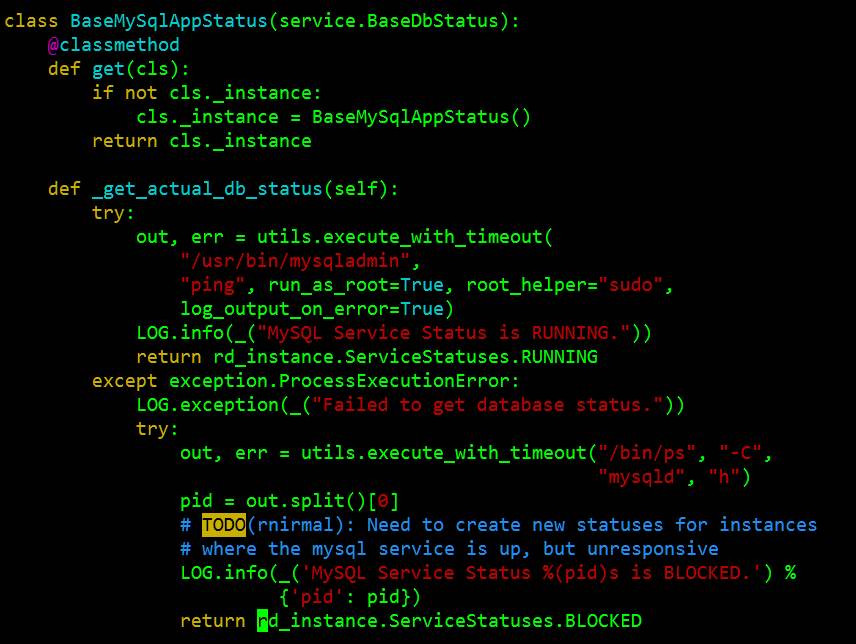
然后数据库实例里面的guest-agent会去读取通过nova注入的trove-guestagent.conf 去连接rpc读取taskmanager发送过来的操作请求。

剩下的一些操作比如创建数据库、创建用户这些都是taskmanager调用数据库实例里面的guest-agent去实现。

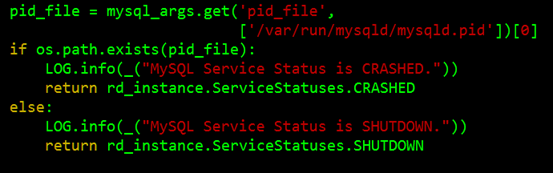
guest-agent对mysql的一些操作实现是在/usr/lib/python2.7/dist-packages/trove/guestagent/datastore/mysql/service\_base.py 他里面包含了

def \_get\_actual\_db\_status() #获取数据库实例状态方法

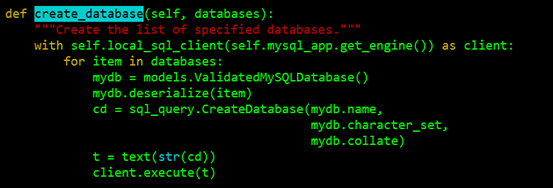
主要是通过调用/usr/bin/mysqladmin ping" 和ps -C mysqld h 去获取数据库实例状态。



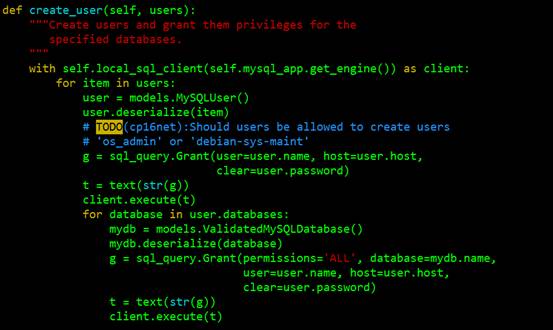
通过判断pid文件是否存在来判断mysql是否shutdown：



def create\_database #创建数据库实例方法：



def create\_user #创建用户并且授权方法：



后面还有删除数据库，删除用户，获取binlog，开始slave、关闭slave等方法。

同时需要注意的是trove创建数据库实例时，会默认为每个数据库实例同时创建一个。

SecGroup-xxx xxx为主机ID的安全组。

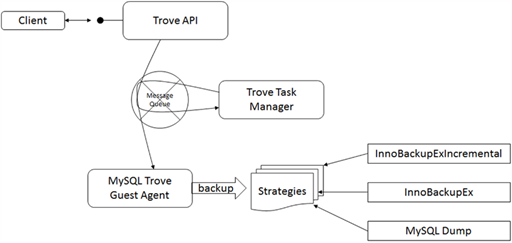
trove默认是不启动root用户的所以在控制台用户选项卡里面用户名称是不能填root的。

需要注意的是上述所有操作都是由trove用户来执行，所以必须要确认的是trove用户拥有sudo权限，否则会失败。

在执行完上述操作前此时数据库状态还是building状态的，那就是 vm 正在启动，创建数据库，创建用户，对用户授权，同步my.cnf配置文件到数据库实例内，重启mysql ，trove-guestagent 发送 rpc 。

给trove-taskmanager，最后检测数据库成功运行后发送 Active 状态消息给 rpc，trove-taskmanager 收到 Active 消息后，不再发送创建数据库消息，而 trove-conductor 同时收到 trove-guestagent Active 消息后，去数据库里更新 trove instance 的状态，在trove list 就可以看见instance Active 的状态了。

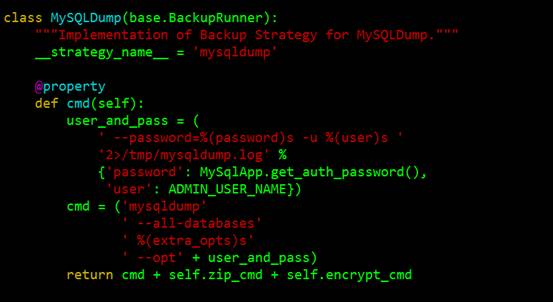
**备份还原**

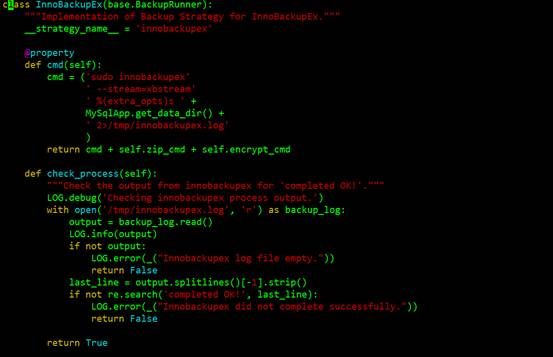


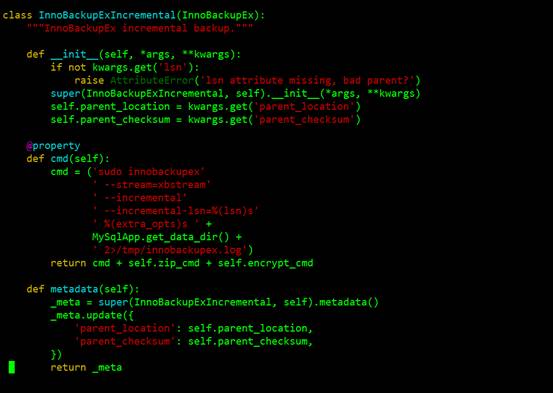
目前trove-guestagent只支持mysql的三种备份方式，一种是传统的mysql Dump方式一种是InnoBackupEx 还有增是InnoBackup的增量备份方式InnoBackupExIncremental。

备份的程序放在/usr/lib/python2.7/dist-packages/trove/guestagent/strategies/backup。

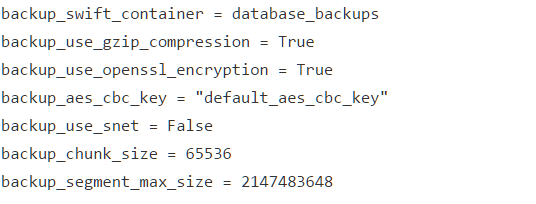
其调用方式也比较简单，就是trove-guestagent.conf里面配置了什么备份方式就调用指定类执行里面的方法，方法内也都是一些软件的命令。





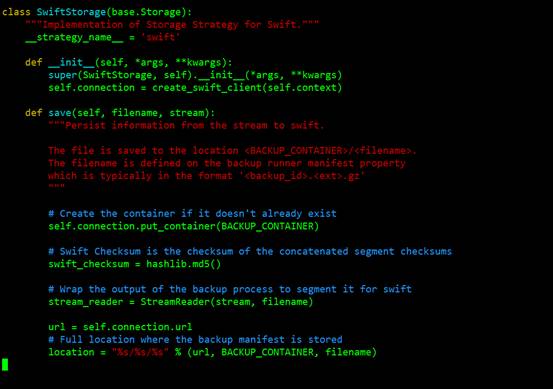


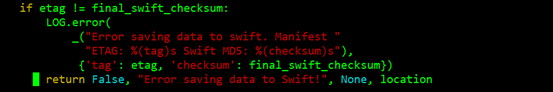
需要注意的是默认不配置是调用Innobackup，备份的日志会存在tmp目录下，备份完成后默认是会存储到swift内。



默认备份在swift内的备份文件夹为database\_backups 、开启压缩、ssl加密，分片等。

调用SwiftStorage类里面的save方法上传到Swift中



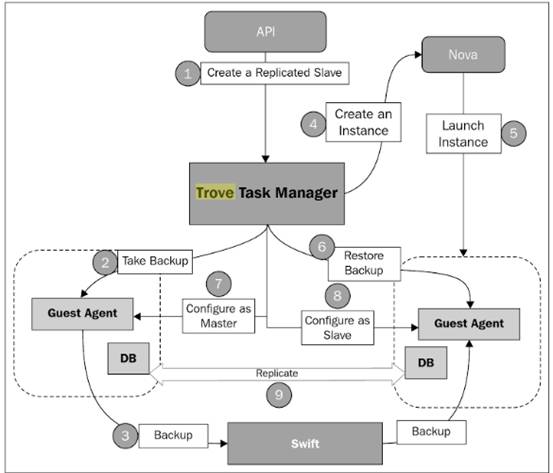


其中会进行文件的校验。

是备份上去的实际上有两个文件，第一个enc文件主要是用来分片使用，第二个文件才是主要的备份文件。

http://img.mp.itc.cn/upload/20170322/1bdbc4820e80490cb4861056b81970d3.png

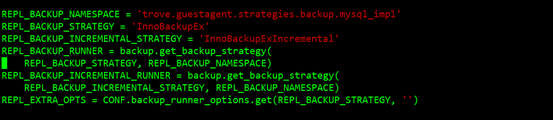
**mysql主从**



trove-master端先将当前数据备份到Swift--->然后taskmanager重新创建个数据库实例------>新创建的数据库实例将刚刚的备份从Swift拉下来根据里面的bin-log里面的GTID进行还原---->建立主从关系---检测创建成功taskmanager删除上传到Swift的备份。

备份前会做个检测，发现以前有备份就调用增量备份的方法节省空间，检测到没有就调用全备的方法。

这里先做个变量定义，定义好增量备份和全备的变量：

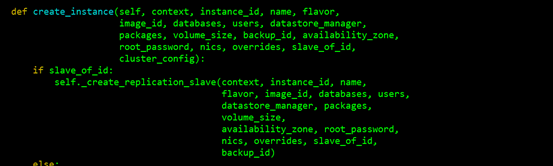


if判断调用全备还是增量备份。



目前trove只支持mysql的主从不支持主主并且还是异步的主从。

创建主从时，创建从同样是调用create\_instance()方法：



只是这里做了个判断，如果传过来了slave\_of\_id就调用\_\_create\_replication\_slave()方法：



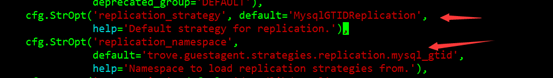
\_\_create\_replication\_slave()方法会去获去备份的ID ,然后继续调用nova创建主机。

接下来操作会交给数据库实例里面的guest-agent进行操作。

guest-agent会先将备份文件从来Swift 下载下来。然后还原。接下来建立主从关系，这里要说明的是trove建立主从关系的方式有两种一种是传统的bin-log的形式，一种的用GTID的形式。

在 /usr/lib/python2.7/dist-packages/trove/common/cfg.py

这个是定义的两个不同的策略：



同时也会调用不同的方法去执行。

当你的配置文件

replication\_strategy = MysqlBinlogReplication

replication\_namespace = trove.guestagent.strategies.replication.mysql\_binlog

调用的是

/usr/lib/python2.7/dist-packages/trove/guestagent/strategies/replication/mysql\_binlog.py

为

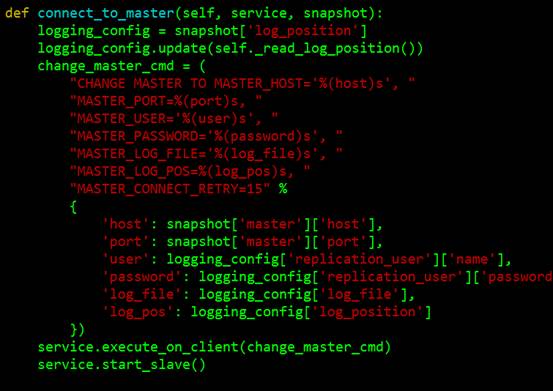
replication\_strategy = MysqlGTIDReplication

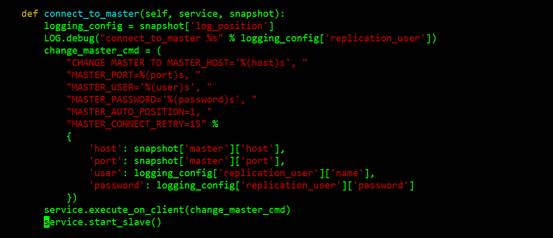
replication\_namespace = trove.guestagent.strategies.replication.mysql\_gtid

调用的是

/usr/lib/python2.7/dist-packages/trove/guestagent/strategies/replication/mysql\_gtid.py

这两个文件有何不同，方法内定义的命令不同：





GTID的概述：

全局事物标识：global transaction identifieds。

GTID事物是全局唯一性的，且一个事务对应一个GTID。一个GTID在一个服务器上只执行一次，避免重复执行导致数据混乱或者主从不一致。GTID用来代替classic的复制方法，不在使用binlog+pos开启复制。而是使用master\_auto\_postion=1的方式自动匹配GTID断点进行复制。MySQL-5.6.5开始支持的，MySQL-5.6.10后开始完善。在传统的slave端，binlog是不用开启的，但是在GTID中，slave端的binlog是必须开启的，目的是记录执行过的GTID（强制）。

**下面介绍一下mysql GTID：**

**GTID的组成部分：**

前面是server\_uuid：后面是一个序列号

例如：server\_uuid：sequence number

7800a22c-95ae-11e4-983d-080027de205a:10

UUID：每个mysql实例的唯一ID，由于会传递到slave，所以也可以理解为源ID。

Sequence number：在每台MySQL服务器上都是从1开始自增长的序列，一个数值对应一个事务。

GTID比传统复制的优势：

* 更简单的实现failover，不用以前那样在需要找log\_file和log\_Pos。
* 更简单的搭建主从复制。
* 比传统复制更加安全。

GTID是连续没有空洞的，因此主从库出现数据冲突时，可以用添加空事物的方式进行跳过。

**GTID的工作原理：**

master更新数据时，会在事务前产生GTID，一同记录到binlog日志中。slave端的i/o 线程将变更的binlog，写入到本地的relay log中。sql线程从relay log中获取GTID，然后对比slave端的binlog是否有记录。如果有记录，说明该GTID的事务已经执行，slave会忽略。如果没有记录，slave就会从relay log中执行该GTID的事务，并记录到binlog。在解析过程中会判断是否有主键，如果没有就用二级索引，如果没有就用全部扫描。

**要点：**

1. slave在接受master的binlog时，会校验master的GTID是否已经执行过（一个服务器只能执行一次）。
2. 为了保证主从数据的一致性，多线程只能同时执行一个GTID。