par-tec
beyond the IT domain

Scaling MySQL with Python

draft2

Roberto Polli - roberto.polli@par-tec.it

Par-Tec Spa - Rome Operation Unit
P.zza S. Benedetto da Norcia, 33
00040, Pomezia *RM* - www.par-tec.it

21-27 July 2015

par-tec
beyond the IT domain

Agenda

- Intro
- MySQL Architecture
- Utilities
  - Administration
  - Export/Import
  - Comparison
  - Replication
- Failover
- Fabric: MySQL Orchestration

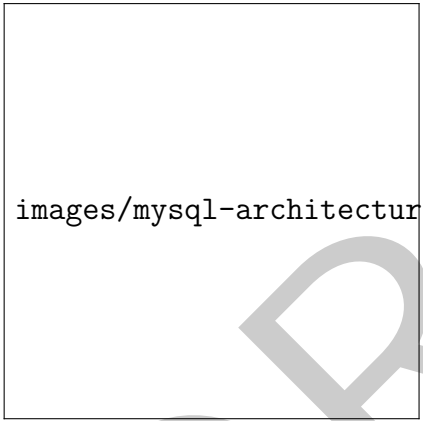
Who? What? Why?

- Manage, replicate, scale MySQL databases with python
- Roberto Polli - Solutions Architect @ par-tec.it. Loves writing in C, Java and Python. Red Hat Certified Engineer and Virtualization Administrator.
- Par-Tec – Proud sponsor of this talk ;) Contributes to various FLOSS.
  Provides expertise in IT Infrastructure & Services and
  Business Intelligence solutions + Vertical Applications for the financial market.

MySQL Architecture

- Frontend (Connection, Caches, Logging)
- Backend (InnoDB Engine)
- Replication

MySQL Architecture



It's a lot of stuff

MySQL Architecture

We should manage and monitor

- Database size: Tables, Indexes, Binary Logs
- Replication inconsistencies
- Failover

Simplify please!

Get the code

```
$ wget http://bit.ly/1CxNuZe -O mysql-utilities-1.6.1.tar.gz
$ tar xf mysql-utilities-1.6.1.tar.gz
$ cd mysql-utilities-1.6.1
$ python setup.py install
```

Utilities

Connectors (drivers)

```
# mysql.connector.django.introspection
if django.VERSION >= (1, 6):
    from django.db.backends import FieldInfo
    if django.VERSION >= (1, 7):
        ...
```

Utilities & Scripts

```
# mysql.utilities.common.replication
if master_innodb_stats != slave_innodb_stats:
    if not pedantic:
        errors.append("WARNING: Innodb settings differ "
                      "between master and slave.")
...
```

Fabric Orchestrator

Roberto Polli - roberto.polli@par-tec.it

par-tec
beyond the IT domain

Single Entrypoint: mysqluc

Start with `mysqluc`

- An entrypoint for all utilities
- Contextual help
- TAB completion

Or call each method separately

- mysqldiskusage
- mysqldbexport / mysqldbimport
- mysqlcompare / mysqldiff
- ...
- mysqlfailover

Roberto Polli - roberto.polli@par-tec.it

Syntax

Define one or more server credentials in the encrypted ~/.mylogin.cnf

```
mysql_config_editor set
    --login-path=client # default used by mysql
    --host=localhost --user=localuser
    --password # (prompted)

mysql # by default uses --login-path=client
```

A SERVER is identified by the string

```
user:password@hostname[:port] # default port 3306
```

or

```
login-path
```

We will use the example sakila database throughout the slide.

Disk usage

A single command to show all disk usage infos (excluded system logs)

```
$ mysqldiskusage --all --server=$SERVER
...
Total database disk usage = 7601892 bytes or 7.25 MB
...
Current binary log file = s-1-bin.000009
...
Total size of binary logs = 231 bytes
```

Export - I

Forget mysqldump and use the following command for a consistent *logical* backup.

```
$ mysqldbexport > data.sql \
    --server=$SERVER
    --all
```

To backup big databases, use InnoDB engine and an InnoDB backup tool!

Import - I

Then import the dump with

```
$ mysqldbimport --server=$SERVER \
    data.sql
```

To provision a new slave we'll use a similar procedure.

Comparing databases - I

To compare databases between servers, use

```
#mysqldbcompare \
    --server1=$MASTER --server2=$SLAVE \
    sakila -a --difftype=SQL \
    --show-reverse --quiet
```

Comparing databases - II

Create the statemets to fix the differences!

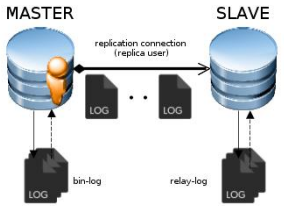
```
mysqldiff \
    --server1=$MASTER --server2=$SLAVE \
    sakila:sakila \ # db name on master:slave
    --changes-for=server2
```

Configuring replication

Replication is *asynchronous* and the agreements are configured on the slave only.

Master

- produces a changelog named binlog;
- grants access to a *replica* user;
- may track slave-updates.



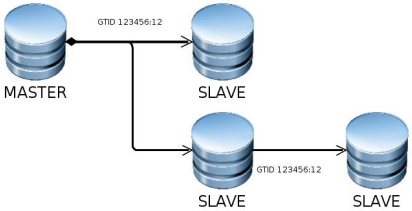
Slave

- connects to the master with the *replica* user
- retireves the binlog and applies the changes;
- `START SLAVE;`

Replication 2.0

MySQL 5.6+ replication is based on Global Transaction ID

- each server has a unique UUID
  eg: `3E11FA47-71CA-11E1-9E33-C80AA9429562`
- every TransactionID becomes global
  eg: `3E11FA47-71CA-11E1-9E33-C80AA9429562:|32|`



If binlog have been purged, you need to import the master database first!

Configuring replication

mysqlreplicate takes care of

- provisioning the replica user on the master;
- configure the slave to point to the master;
- start loading the first available transaction in bin-logs;

```
mysqlreplicate  --master=$MASTER --slave=$SLAVE \
        --rpl-user=repl:rpass \
        -b

# master on 192.168.1.1: ... connected.
# slave on 192.168.1.2: ... connected.
# Checking for binary logging on master...
# Setting up replication...
# ...done.
```

Configuring replication - II

mysqldbexport can be used to provision a new slave!

- issue a `RESET MASTER;` to cleannup previous settings;
- add `--rpl=master` to create replica infos in the sql;
- add `--export=both` to store both schema and data;

```
# pre-import.sql
-- ignore previous changes
-- and trust the backup
STOP SLAVE;
RESET MASTER;
```

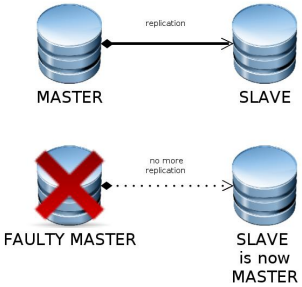
```
$ mysqldbexport > data.sql \
--server=$MASTER \
--rpl-user=repl:rpass \
--export=both \
--rpl=master --all
```

Discovering replication

```
$ mysqlrplshow --master=$MASTER \
    --discover-slaves-login=root:root
# master on s-1.docker: ... connected.
# |Finding slaves| for master: s-1.docker:3306
# Replication Topology Graph
s-1.docker:3306 (MASTER)
   |
   +--- s-3.docker:3306 - (SLAVE)
   |
   +--- s-4.docker:3306 - (SLAVE)
```

Failover Basics

A replicated infrastructure can be made Higly Available.



In case of fault you should:

- *promove* your slave!
- reconfigure the others to point there
- disable the master
- eventually switch the ip-address

Failover - I

mysqlfailover takes care of that, and can even discover your replication topology!

```
$ mysqlfailover --master=$MASTER \
    --discover-slaves-login=root:password \
    --candidates=$SLAVE1,$SLAVE2 \
    --exec-before=/pre-fail.sh \
    --exec-after=/post-fail.sh
```

mysqlfailover supports a lot of parameters! Read them carefully and test thoroughly your solution

Failover - II

Run mysqlfailover on an existing infrastructure!

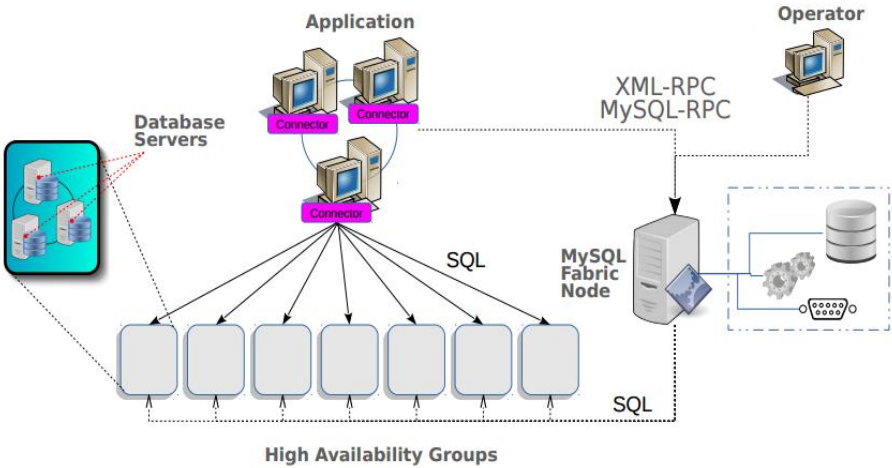
```
$ mysqlfailover --master=$MASTER \
 --discover-slaves-login=root:root
# Discovering slaves for master at s-1.docker:3306
# Discovering slave at s-3.docker:3306
# Found slave: s-3.docker:3306
# Discovering slave at s-4.docker:3306
# Found slave: s-4.docker:3306
# Checking privileges.
...
```

Failover - III

Run mysqlfailover on an existing infrastructure!

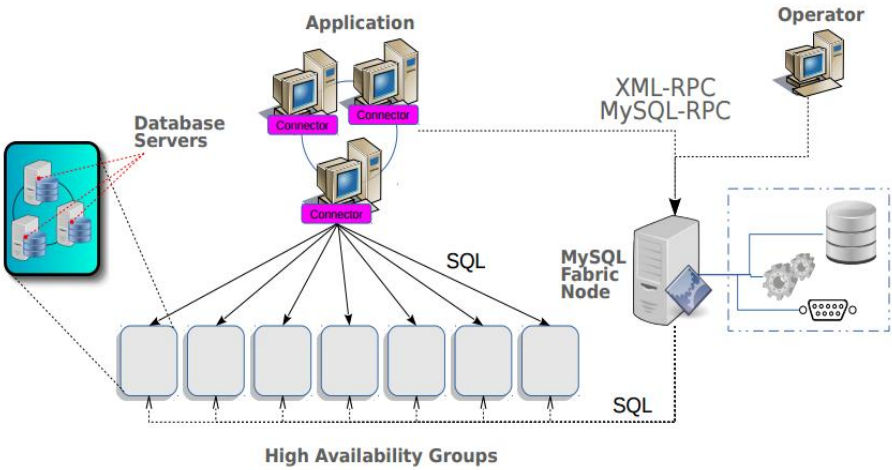
```
MySQL Replication Failover Utility
Failover Mode = auto     Next Interval = Sun Apr 12 14:32:40 2015
...
Replication Health Status
+-------------+-------+---------+--------+------------+---------+
| host        | port  | role    | state  | gtid_mode  | health  |
+-------------+-------+---------+--------+------------+---------+
| s-1.docker  | 3306  | MASTER  | UP     | ON         | OK      |
| s-3.docker  | 3306  | SLAVE   | UP     | ON         | OK      |
| s-4.docker  | 3306  | SLAVE   | UP     | ON         | OK      |
+-------------+-------+---------+--------+------------+---------+
```

Fabric - I



Fabric is a python framework for managing, replicating, sharding and scaling mysql

Fabric HLA - II



Fabric Setup

Configure `/etc/mysql/fabric.cfg` setting, then setup

```
# Create fabric database and
# configure endpoint properties
mysqlfabric manage setup --param=storage.user=fabric

# Startup and check if ok
mysqlfabric manage start
mysqlfabric manage ping
```

Fabric Groups - create add

Create an High Availability group and add one or more servers

```
# add servers to fabric
mysqlfabric group create $HA
mysqlfabric group add $HA $SERVER1
...
mysqlfabric group add $HA $SERVERX
```

Fabric Groups - lookup

Show groups

```
[root@fabric /]# mysqlfabric group lookup_groups
Fabric UUID:  5ca1ab1e-a007-feed-f00d-cab3fe13249e
Time-To-Live: 1

group_id description failure_detector master_uuid
-------- ----------- ---------------- -----------
      ha         None                1 f0ce9615...
```

Fabric Groups - promote, activate

Now start the game

```
# Set one server as master...
$ mysqlfabric group \
    promote $HA \
     --slave_id f0ce9615-df69-11e4-b909-0242ac11000a

# .. and enable monitoring and failover
$ mysqlfabric group activate $HA
```

Fabric Groups - health

Use hea

```
# and check if the group is fine
$ mysqlfabric group heath $HA
       uuid is_alive    status ... io_error sql_error
----------- -------- --------- --- -------- ---------
da42f6b1...        1 SECONDARY        False     False
f0ce9615...        1   PRIMARY        False     False
```

Fabric in the Cloud

Fabric can provision new servers via Openstack API.

```
$ mysqlfabric server create ...
```

Initialize new servers with

```
$ mysqlfabric server clone $GROUP $TARGET
```

This will initialize TARGET from the GROUP's master without attaching TARGET to the group nor starting the replica.

Fabric in the Cloud

We implemented a Docker API provider

```
# mysql.fabric.providers.dockerprovider
...
class MachineManager(AbstractMachineManager):
    """Manage a Docker Machine.
    """
    def create(self, parameters, wait_spawning):
        ...
    def destroy(self, machine):
        ...
```

Wrap Up

- Use MySQL Utilities for custom replication and failover setup
- Mash-up the underlying modules
- Use Fabric for standard, highly-available, master-slave topologies
- Try Fabric for provisioning and cloning servers

That's all folks!

Thank you for the attention!
Roberto Polli - roberto.polli@par-tec.it