pg_chameleon

MySQL to PostgreSQL replica made easy

Federico Campoli

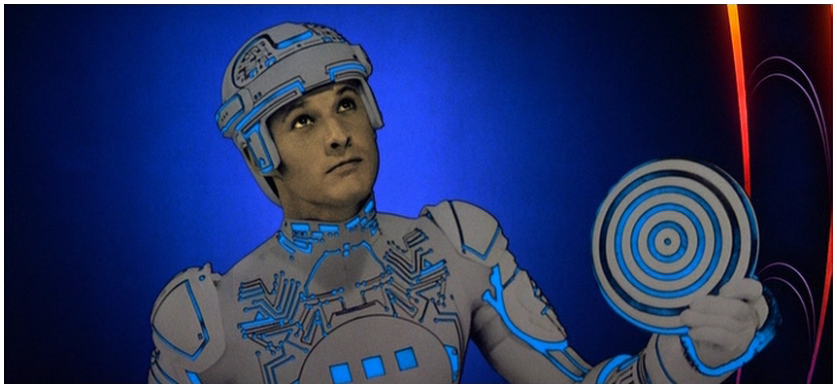
Transferwise

PGCon, Ottawa
01 Jun 2018

http://www.pgdba.org

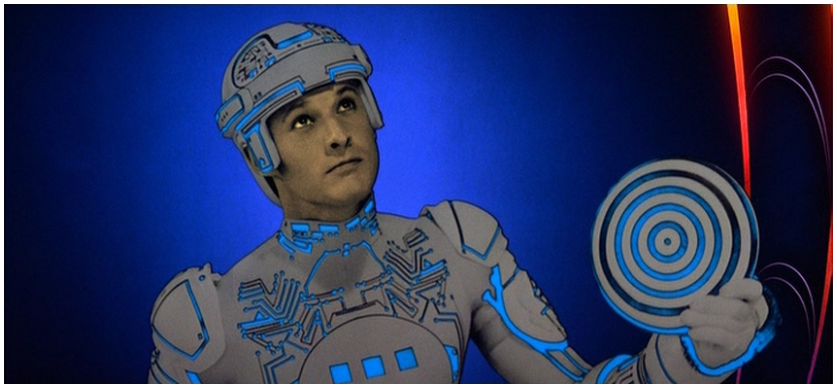
@4thdoctor_scarf

Few words about the speaker



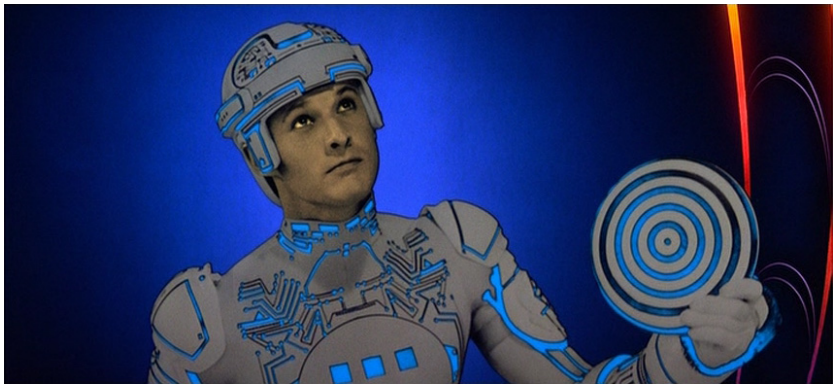
- Born in 1972
- Passionate about IT since 1982
- mostly because of the TRON movie

Few words about the speaker



- Born in 1972
- Passionate about IT since 1982
- mostly because of the TRON movie
- Joined the Oracle DBA secret society in 2004
- In love with PostgreSQL since 2006

Few words about the speaker



- Born in 1972
- Passionate about IT since 1982
- mostly because of the TRON movie
- Joined the Oracle DBA secret society in 2004
- In love with PostgreSQL since 2006
- Devrim PostgreSQL tattoo's copycat
- Works at Transferwise as Data Engineer

Disclaimer

- I'm not a developer
- I'm a DBA...

Disclaimer

- I'm not a developer
- I'm a DBA...which means being hated by everybody and hating everybody

Disclaimer

- I'm not a developer
- I'm a DBA...which means being hated by everybody and hating everybody
- So, to put things in the right perspective...

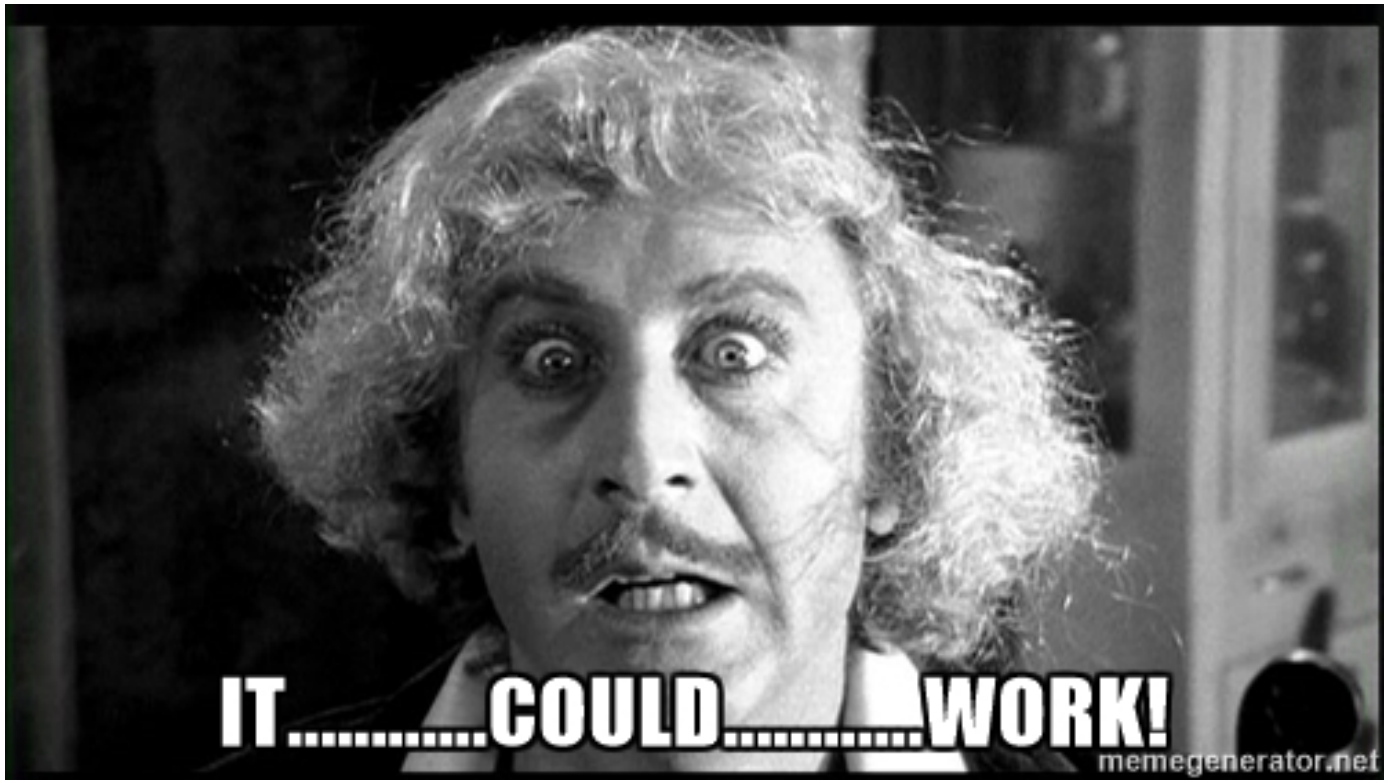
Disclaimer

- I'm not a developer
- I'm a DBA...which means being hated by everybody and hating everybody
- So, to put things in the right perspective...I use tabs

Palpatine

Table of contents

History



The beginnings

Years 2006/2012

neo_my2pg.py

- I wrote the script because of a struggling phpbb on MySQL
- The database migration was successful
- However phpbb didn't work very well with PostgreSQL.[1]

[1] Opening a new connection for each query is not the smartest thing to do.

The beginnings

Years 2006/2012

neo_my2pg.py

- I wrote the script because of a struggling phpbb on MySQL
- The database migration was successful
- However phpbb didn't work very well with PostgreSQL.[1]
- The script is written in python 2.6
- It's a monolith script
- And it's slow, very slow

[1] Opening a new connection for each query is not the smartest thing to do.

The beginnings

Years 2006/2012

neo_my2pg.py

- I wrote the script because of a struggling phpbb on MySQL
- The database migration was successful
- However phpbb didn't work very well with PostgreSQL.[1]
- The script is written in python 2.6
- It's a monolith script
- And it's slow, very slow
- It's a good checklist for things to avoid when coding

https://github.com/the4thdoctor/neo_my2pg

[1] Opening a new connection for each query is not the smartest thing to do.

I'm not scared of using the ORMs

Years 2013/2015
First attempt of pg_chameleon

- Developed in Python 2.7
- Used SQLAlchemy for extracting the MySQL's metadata
- Proof of concept only
- It was built during the years of the life on a roller coaster[2]
- Therefore it was a just a way to discharge frustration

[2] Recording available here: http://www.pgbrighton.uk/post/backup_recovery/

I'm not scared of using the ORMs

Years 2013/2015
First attempt of pg_chameleon

- Developed in Python 2.7
- Used SQLAlchemy for extracting the MySQL's metadata
- Proof of concept only
- It was built during the years of the life on a roller coaster[2]
- Therefore it was a just a way to discharge frustration
- Abandoned after a while
- SQLAlchemy's limitations were frustrating as well (see slide 3)
- And pgloader did the same job much much better

[2] Recording available here: http://www.pgbrighton.uk/post/backup_recovery/

pg_chameleon reborn

Year 2016

- I needed to replicate the data data from MySQL to PostgreSQL
- http://tech.transferwise.com/scaling-our-analytics-database/

pg_chameleon reborn

Year 2016

- I needed to replicate the data data from MySQL to PostgreSQL
- http://tech.transferwise.com/scaling-our-analytics-database/
- The amazing library python-mysql-replication allowed me build a proof of concept
- Evolved later in pg_chameleon 1.x

Kudos to the python-mysql-replication team!

https://github.com/noplay/python-mysql-replication

pg_chameleon 1.x

- Developed on the London to Brighton commute
- Released as stable the 7th May 2017
- Followed by 8 bugfix releases

pg_chameleon 1.x

- Developed on the London to Brighton commute
- Released as stable the 7th May 2017
- Followed by 8 bugfix releases
- Compatible with CPython 2.7/3.3+
- No more SQLAlchemy
- The MySQL driver changed from MySQLdb to PyMySQL
- Command line helper
- Supports type override on the fly (Danger!)
- Installs in virtualenv and system wide via pypi
- Can detach the replica for minimal downtime migrations

pg_chameleon versions 1's limitations

- All the affected tables are locked in read only mode during the init_replica process
- During the init_replica the data is not accessible

pg_chameleon versions 1's limitations

- All the affected tables are locked in read only mode during the init_replica process
- During the init_replica the data is not accessible
- The tables for being replicated require primary keys
- No daemon, the process always stays in foreground
- Single schema replica
- One process per each schema
- Network inefficient

pg_chameleon versions 1's limitations

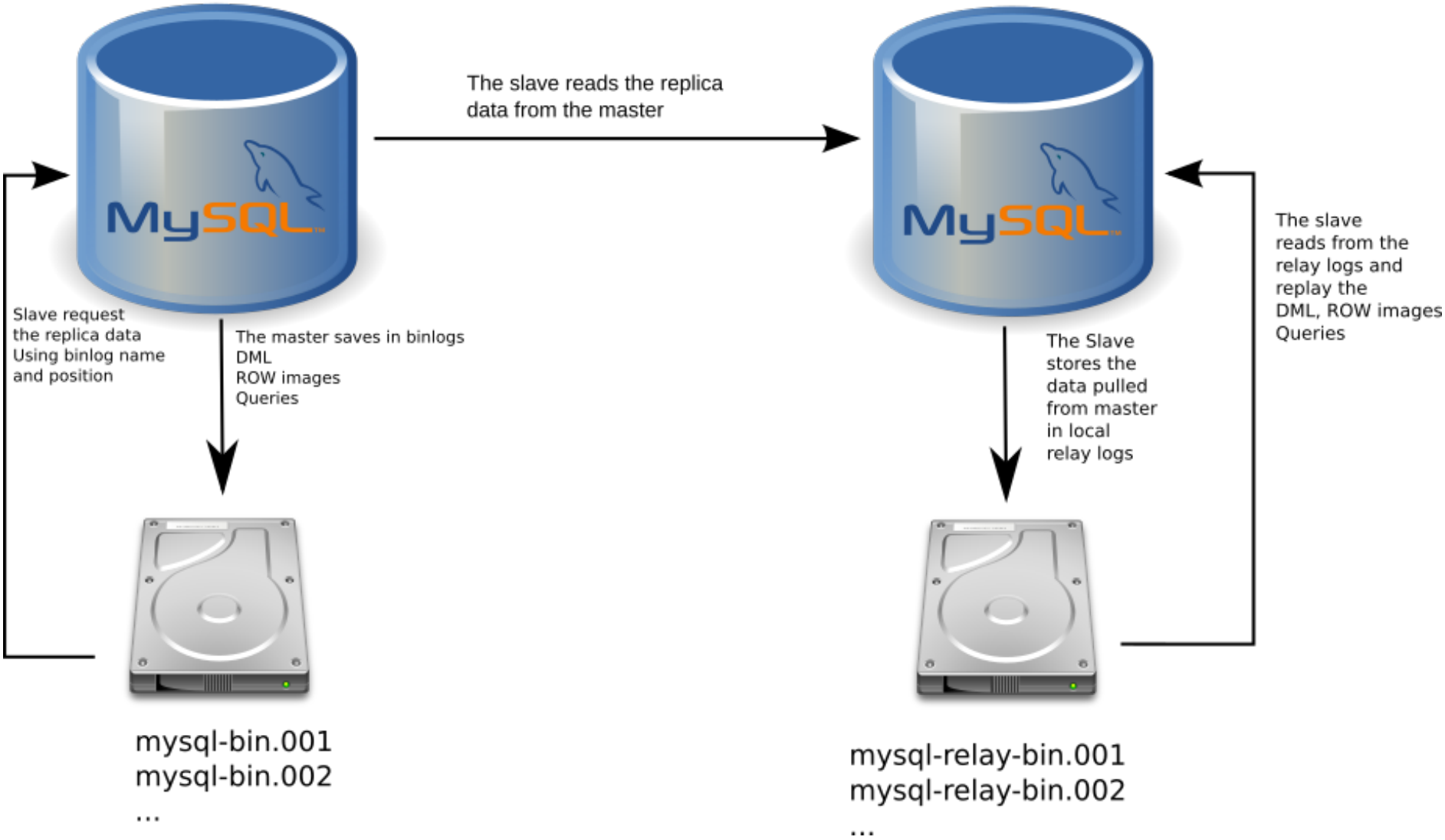
- All the affected tables are locked in read only mode during the init_replica process
- During the init_replica the data is not accessible
- The tables for being replicated require primary keys
- No daemon, the process always stays in foreground
- Single schema replica
- One process per each schema
- Network inefficient
- Read and replay not concurrent with risk of high lag
- The optional threaded mode very inefficient and fragile
- A single error in the replay process and the replica is broken

MySQL Replica in a nutshell

MySQL Replica

- The MySQL replica is logical
- When the replica is enabled the data changes are stored in the master's binary log files
- The slave gets from the master's binary log files
- The slave saves the stream of data into local relay logs
- The relay logs are replayed against the slave

MySQL Replica



Log formats

MySQL have three ways of storing the changes in the binary logs.

- STATEMENT: It logs the statements which are replayed on the slave. It's the best solution for the bandwidth. However, when replaying statements with not deterministic functions this format generates different values on the slave (e.g. using an insert with a column autogenerated by the uuid function).
- ROW: It's deterministic. This format logs the row images.
- MIXED takes the best of both worlds. The master logs the statements unless a not deterministic function is used. In that case it logs the row image.

All three formats always log the DDL query events.
The python-mysql-replication library and therefore pg_chameleon, require the ROW format to work properly.

A chameleon in the middle

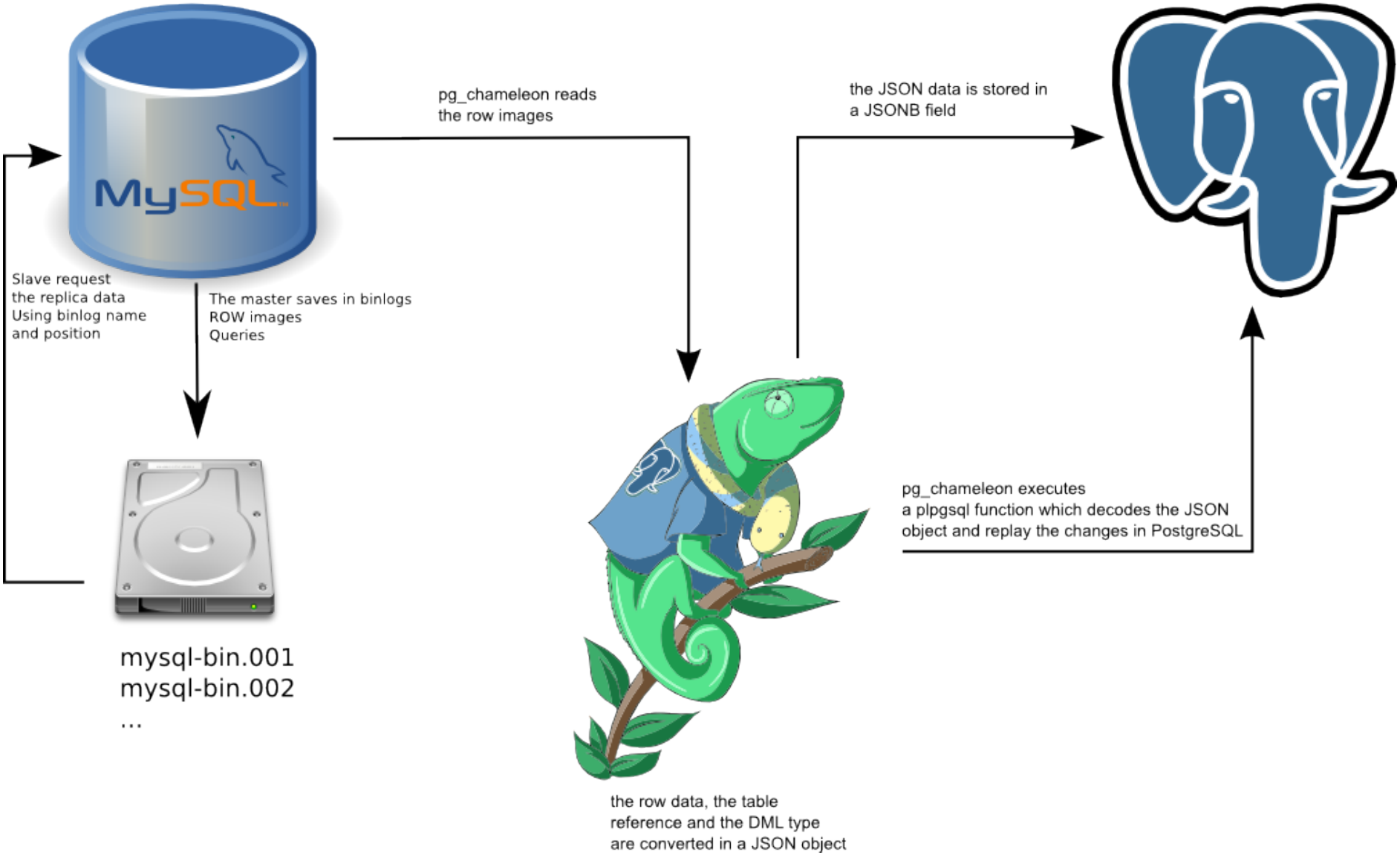
pg_chameleon

pg_chameleon mimics a mysql slave's behaviour

- It performs the initial load for the replicated tables
- It connects to the MySQL replica protocol
- It stores the row images into a PostgreSQL table
- A plpgSQL function decodes the rows and replay the changes
- It can detach the replica for minimal downtime migrations

PostgreSQL acts as relay log and replication slave

MySQL replica + pg_chameleon



Why pg_chameleon?

PostgreSQL supports federated tables and tools like pgloader can migrate easily the data from MySQL.
Why I should setup a replica instead?

Why pg_chameleon?

PostgreSQL supports federated tables and tools like pgloader can migrate easily the data from MySQL.
Why I should setup a replica instead?

- I'm happy with MySQL, I just need a database for analytics

Why pg_chameleon?

PostgreSQL supports federated tables and tools like pgloader can migrate easily the data from MySQL.
Why I should setup a replica instead?

- I'm happy with MySQL, I just need a database for analytics
- I want to migrate to PostgreSQL with minimal downtime

Why pg_chameleon?

PostgreSQL supports federated tables and tools like pgloader can migrate easily the data from MySQL.
Why I should setup a replica instead?

- I'm happy with MySQL, I just need a database for analytics
- I want to migrate to PostgreSQL with minimal downtime
- My ginormous queries will kill the network if I'd use the foreign data wrapper

Why pg_chameleon?

PostgreSQL supports federated tables and tools like pgloader can migrate easily the data from MySQL.
Why I should setup a replica instead?

- I'm happy with MySQL, I just need a database for analytics
- I want to migrate to PostgreSQL with minimal downtime
- My ginormous queries will kill the network if I'd use the foreign data wrapper
- I can't get rid of MySQL but I need that super cool PostgreSQL's feature

pg_chameleon 2.0 #1

- Developed at the pgconf.eu 2017 and on the commute
- Released as stable the 1st of January 2018
- Compatible with python 3.3+
- Installs in virtualenv and system wide via pypi
- Replicates multiple schemas from a single MySQL into a target PostgreSQL database
- Conservative approach to the replica. Tables which generate errors are automatically excluded from the replica
- Daemonised replica process with two distinct subprocesses, for concurrent read and replay

pg_chameleon 2.0 #2

- Soft locking replica initialisation. The tables are locked only during the copy.
- Rollbar integration for a simpler error detection and messaging
- Experimental support for the PostgreSQL source type
- The tables are loaded in a separate schema which is swapped with the existing.
  This approach requires more space but it makes the init a replica virtually painless, leaving the old data accessible until the init_replica is complete.
- The DDL are translated in the PostgreSQL dialect keeping the schema in sync with MySQL automatically

Version 2.0's limitations

- Tables for being replicated require primary or unique keys
- When detaching the replica the foreign keys are created always ON DELETE/UPDATE RESTRICT
- The source type PostgreSQL supports only the init_replica process

Replica initialisation

The replica initialisation follows the same workflow as stated on the mysql online manual.

- Flush the tables with read lock
- Get the master's coordinates
- Copy the data
- Release the locks

However...

Replica initialisation

The replica initialisation follows the same workflow as stated on the mysql online manual.

- Flush the tables with read lock
- Get the master's coordinates
- Copy the data
- Release the locks

However...
pg_chameleon flushes the tables with read lock one by one. The lock is held only during the copy.

The log coordinates are stored in the replica catalogue along the table's name and used by the replica process to determine whether the table's binlog data should be used or not.

The replica starts inconsistent and gains consistency over time.

Fallback on failure

The data is pulled from mysql using the CSV format in slices. This approach prevents the memory overload.

Once the file is saved then is pushed into PostgreSQL using the COPY command. However...

Fallback on failure

The data is pulled from mysql using the CSV format in slices. This approach prevents the memory overload.

Once the file is saved then is pushed into PostgreSQL using the COPY command. However...

- COPY is fast but is single transaction
- One failure and the entire batch is rolled back
- If this happens the procedure loads the same data using the INSERT statements
- Which can be very slow
- The process attempts to clean the NUL markers which are allowed by MySQL
- If the row still fails on insert then it's discarded

Replica in action

MySQL configuration

The mysql configuration file is usually stored in /etc/mysql/my.cnf
To enable the binary logging find the section [mysqld] and check that the following parameters are set.

```
binlog_format= ROW
log-bin = mysql-bin
server-id = 1
binlog-row-image = FULL
```

MySQL user for replica

Setup a replication user on MySQL

```
CREATE USER usr_replica ;
SET PASSWORD FOR usr_replica=PASSWORD('replica');
GRANT ALL ON sakila.* TO 'usr_replica';
GRANT RELOAD ON *.* to 'usr_replica';
GRANT REPLICATION CLIENT ON *.* to 'usr_replica';
GRANT REPLICATION SLAVE ON *.* to 'usr_replica';
FLUSH PRIVILEGES;
```

In our example we are using the sakila test database.
https://dev.mysql.com/doc/sakila/en/

PostgreSQL setup

Add an user on PostgreSQL capable to create schemas and relations in the destination database

```
CREATE USER usr_replica WITH PASSWORD 'replica';
CREATE DATABASE db_replica WITH OWNER usr_replica;
```

Install pg_chameleon

Install pg_chameleon and create the configuration files

```
pip install pip --upgrade
pip install pg_chameleon
chameleon set_configuration_files
cd ~/.pg_chameleon/configuration
cp config-example.yml default.yml
```

Edit the file default.yml setting the correct values for connection and source.

Configure global settings in default.yaml

PostgreSQL Connection

```
pg_conn:
    host: "localhost"
    port: "5432"
    user: "usr_replica"
    password: "replica"
    database: "db_replica"
    charset: "utf8"
```

Configure global settings in default.yaml

PostgreSQL Connection

```
pg_conn:
    host: "localhost"
    port: "5432"
    user: "usr_replica"
    password: "replica"
    database: "db_replica"
    charset: "utf8"
```

Rollbar configuration

```
rollbar_key: '<rollbar_long_key>'
rollbar_env: 'pgcon-demo'
```

Configure global settings in default.yaml

PostgreSQL Connection

```
pg_conn:
    host: "localhost"
    port: "5432"
    user: "usr_replica"
    password: "replica"
    database: "db_replica"
    charset: "utf8"
```

Rollbar configuration

```
rollbar_key: '<rollbar_long_key>'
rollbar_env: 'pgcon-demo'
```

Type override (optional)

```
type_override:
    "tinyint(1)":
        override_to: boolean
        override_tables:
            - "*"
```

Configure the mysql source

```
sources:
  mysql:
    db_conn:
      host: "localhost"
      port: "3306"
      user: "usr_replica"
      password: "replica"
      charset: 'utf8'
      connect_timeout: 10
```

Configure the mysql source

```
sources:
  mysql:
    db_conn:
      host: "localhost"
      port: "3306"
      user: "usr_replica"
      password: "replica"
      charset: 'utf8'
      connect_timeout: 10


    schema_mappings:
      sakila: loxodonta_africana
```

Configure the mysql source

```
sources:
  mysql:
    db_conn:
      host: "localhost"
      port: "3306"
      user: "usr_replica"
      password: "replica"
      charset: 'utf8'
      connect_timeout: 10


    schema_mappings:
      sakila: loxodonta_africana



    limit_tables:
    skip_tables:
    grant_select_to:
      - usr_readonly
    lock_timeout: "120s"
    my_server_id: 100
    replica_batch_size: 10000
    replay_max_rows: 10000
    batch_retention: '1 day'
    copy_max_memory: "300M"
    copy_mode: 'file'
    out_dir: /tmp
    sleep_loop: 1
    on_error_replay: continue
    on_error_read: continue
    auto_maintenance: "1 day"
    type: mysql
```

Add the source and initialise the replica

Add the source mysql and initialise the replica for it. We are using debug in order to get the logging on the console.

```
chameleon create_replica_schema --debug
chameleon add_source --config default --source mysql --debug
chameleon init_replica --config default --source mysql --debug
```

Start the replica

Start the replica process

```
chameleon start_replica --config default --source mysql
```

Start the replica

Start the replica process

```
chameleon start_replica --config default --source mysql
```

Show the replica status

```
chameleon show_status --config default --source mysql
```

```
(testcham) thedoctor@tardis:~$ chameleon.py show_status --source mysql
  Source id  Source name    Type    Status    Consistent    Read lag    Last read           Replay lag    Last replay
-----------  -------------  ------  --------  ------------  ----------  ------------------  ------------  ------------------
          1  mysql          mysql   running   Yes           00:01:29    2017-12-10 21:57:34  00:00:00      2017-12-10 21:57:34

== Schema mappings ==
Origin schema    Destination schema
---------------  --------------------
sakila           sch_sakila

== Replica status ==
--------------------  -----
Tables not replicated     0
Tables replicated        18
All tables               18
Replayed rows         97272
Replayed DDL             18
Skipped rows              3
--------------------  -----
```

Time for a demo

Demo!

Obviously the demo will fail miserably and you will hate this project forever.

Lessons learned

Strictness is an illusion. MySQL doubly so

MySQL's lack of strictness is not a mystery.
The funny way the default with NOT NULL is managed by MySQL can break the replica.

Therefore any field with NOT NULL added after the initialisation are created always as NULLable in PostgreSQL.

The DDL. A real pain in the back

I initially tried to use sqlparse for tokenising the DDL emitted by MySQL.
Unfortunately didn't worked as I expected.

The DDL. A real pain in the back

I initially tried to use sqlparse for tokenising the DDL emitted by MySQL.
Unfortunately didn't worked as I expected.
So I decided to use the regular expressions.

```
Some people, when confronted with a problem,
think "I know, I'll use regular expressions."
Now they have two problems.

-- Jamie Zawinski
```

The DDL. A real pain in the back

I initially tried to use sqlparse for tokenising the DDL emitted by MySQL.
Unfortunately didn't worked as I expected.
So I decided to use the regular expressions.

```
Some people, when confronted with a problem,
think "I know, I'll use regular expressions."
Now they have two problems.

-- Jamie Zawinski
```

- MySQL even in ROW format emits the DDL as statements
- The class sql_token uses the regular expressions to tokenise the DDL
- The tokenised data is used to build the DDL in the PostgreSQL dialect

Wrap up

To boldly go where no chameleon has gone before

Short team goals, version 2.0

- Re sync automatically the tables when they error on replay
- Improve the replay speed and cpu efficiency
- GTID support for MySQL source

Medium term goals version 2.1

- Parallel copy and index creation in order to speed up the init_replica process
- Logical replica from PostgreSQL
- Improve the default column handling

Igor, the green little guy

The chameleon logo has been developed by Elena Toma, a talented Italian Lady.

https://www.facebook.com/Tonkipapperoart/

The name Igor is inspired by Martin Feldman's Igor portraited in Young Frankenstein movie.

Feedback please!

Please report any issue on github and follow pg_chameleon on twitter for the announcements.

https://github.com/the4thdoctor/pg_chameleon


@pg_chameleon

Did you say hire?

TransferWise

WE ARE HIRING!

https://transferwise.com/jobs/

That's all folks!

Thank you for listening!

Any questions?
Please be very basic, I'm just an electrician after all.

Image credits

- Palpatine,Dr. Evil disclaimer,It could work. Young Frankenstein source memegenerator
- MySQL Image source, WikiCommons
- Hard Disk image, source WikiCommons
- Tron image, source Tron Wikia
- Twitter icon, source Open Icon Library
- The PostgreSQL logo, copyright the PostgreSQL global development group
- Boromir get rid of mysql, source imgflip
- Morpheus, source imgflip
- Keep calm chameleon, source imgflip
- The dolphin picture - Copyright artnoose
- Perseus, Framed - Copyright Federico Campoli
- Pinkie Pie that's all folks, Copyright by dan232323, used with permission
- Doom, source RetroPie

License

pg_chameleon

MySQL to PostgreSQL replica made easy

Federico Campoli

Transferwise

PGCon, Ottawa
01 Jun 2018

http://www.pgdba.org

@4thdoctor_scarf