Few words about the speaker

- Born in 1972
- **Passionate** about IT since 1982
- Joined the Oracle DBA secret society in 2004
- In love with PostgreSQL since 2006
- PostgreSQL tattoo on the right shoulder
- Works at Loxodata as PostgreSQL Consultant
Loxodata

- A company built on 3 pillars

postgresql  devops  cloud
Loxodata

• A comprehensive service offer

architecture  consulting  training
administration  audit  support
Agenda

• History
• The MySQL Replica
• pg_chameleon v2.0
• The replica in action
• Demo
• Wrap up
History

How the project began
The beginning

- Years 2006/2012 neo_my2pg.py
- I wrote the script because of a struggling phpbb on MySQL
- 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](https://github.com/the4thdoctor/neo_my2pg)
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 just a way to discharge frustration
- Abandoned after a while
- SQLAlchemy's limitations were frustrating as well
- And pgloader did the same job much much better
pg_chameleon reborn

• Year 2016
  ● I needed to replicate the data from MySQL to PostgreSQL
  ● The amazing library python-mysql-replication allowed me build a proof of concept
  ● Which evolved later in pg_chameleon 1.x

• Kudos to the python-mysql-replication team!
• [https://github.com/noplay/python-mysql-replication](https://github.com/noplay/python-mysql-replication)
pg_chameleon v1.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 v1.x 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
The MySQL replica

Do I really need to do that?
MySQL replica in a nutshell

- 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 in a nutshell

The slave reads the replica data from the master

Slave request the replica data
Using binlog name and position

The master saves in binlogs
DML
ROW Images
Queries

mysql-bin.001
mysql-bin.002
...

mysql-relay-bin.001
mysql-relay-bin.002
...

The slave reads from the relay logs and replay the DML, ROW Images Queries

The slave stores the data pulled from master in local relay logs
The log formats

- MySQL can store the changes in the binary logs in three different formats
  - 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 used by pg_chameleon require the ROW format to work properly.
pg_chameleon v2.0

Overview of the stable release
A chameleon in the middle

- 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
A chameleon in the middle

- Slave request the replica data using binlog name and position
- The master saves binlogs
- Queries

mysql-bin.001
mysql-bin.002
...

pg_chameleon reads the row images

the JSON data is stored in a JSONB field

pg_chameleon executes a plpgsql function which decodes the JSON object and replay the changes in PostgreSQL

the row data, the table reference and the DML type are converted in a JSON object
pg_chameleon v2.0

- 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 v2.0

- 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
- MySQL GTID support for switch across the replica cluster without need for init_replica
pg_chameleon v2.0 limitations

- The 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
- Problems on Amazon RDS with the json data type
- No support for MariaDB’s GTID
The replica in action

Let's configure the replica for our example
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 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
MySQL configuration

The mysql configuration file on linux 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 database user

Setup a MySQL database user for the replica

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;
Setup a PostgreSQL database user and a database owned by the freshly created user

```
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.

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

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

```
pg_conn:
    host: "localhost"
    port: "5432"
    user: "usr_replica"
    password: "replica"
    database: "db_replica"
    charset: "utf8"
rollbar_key:
    '<rollbar_long_key>'
rollbar_env: 'demo'
type_override:
    "tinyint(1)"
    override_to: boolean
    override_tables: 
```

MySQL source settings

sources:
mysql:
db_conn:
  host: "localhost"
  port: "3306"
  user: "usr_replica"
  password: "replica"
  charset: 'utf8'
  connect_timeout: 10
schema_mappings:
  sakila: db_sakila
limit_tables:
skip_tables:
grant_select_to:
  - usr_readonly
lock_timeout: "120s"
my_server_id: 100
MySQL source settings

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"
gtid_enable: No
type: mysql
MySQL source settings

skip_events:
insert:
  - sakila.author #skips inserts on the table sakila.author
delete:
  - sakila2 #skips deletes on schema sakila2
update:
Register the source and init 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 for the source

```bash
chameleons start_replica --config default --source mysql
chameleons show_status --config default --source mysql
```

<table>
<thead>
<tr>
<th>Source id</th>
<th>Source name</th>
<th>Type</th>
<th>Status</th>
<th>Consistent</th>
<th>Read lag</th>
<th>Last read</th>
<th>Replay lag</th>
<th>Last replay</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>mysql</td>
<td>mysql</td>
<td>running</td>
<td>Yes</td>
<td>00:01:29</td>
<td>2017-12-10 21:57:34</td>
<td>00:00:00</td>
<td>2017-12-10 21:57:34</td>
</tr>
</tbody>
</table>

== 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
Demonstration

Which will fail miserably and you’ll hate this project forever
Wrap up

Lesson learned, future development and thanks
NOT NULL and no default make slonik cry

The way the implicit defaults with NOT NULL are managed by MySQL can break the replica on PostgreSQL.

Therefore any field with NOT NULL added after the init_replica is created always as NULLable in PostgreSQL.
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
Future development

The development of the version 2.1 is started

Things that will likely appear in the not so distant future

- Parallel copy and index creation in order to speed up the init_replica process
- Logical replica from PostgreSQL
- Improve the default column handling
- Locale support
- Auto sync for the tables removed from the replica
Igor, the little green 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

Twitter: @pg_chameleon
We are hiring!

Please send your CV to jobs@loxodata.com
Thank you for listening!