pg_chameleon

Replicare MySQL in PostgreSQL con semplicità

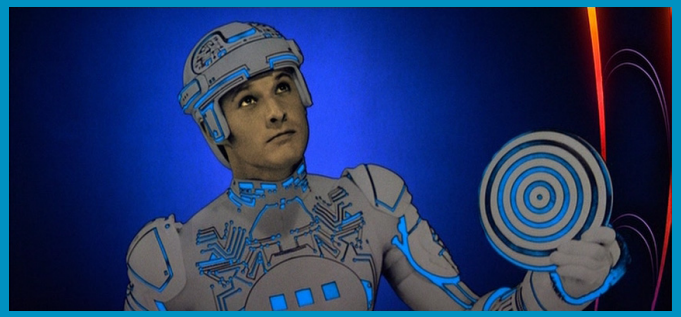
Federico Campoli

Kamedata

PGDay Italia, Bologna 17 Maggio 2019

Due parole su di me

KAMEDATA
DEVOPS AND DATA ENGINEERING



- Classe 1972
- Appassionato di informatica dal 1982
- grazie al film TRON
- Entrato nella setta dei DBA Oracle nel 2004
- Innamorato di PostgreSQL dal 2006
- Tatuaggio PostgreSQL logo sulla spalla destra
- Consulente freelance devops e data engineering

Whether you need a simple audit a tailored training or support for your infrastructure, we can help you to improve.

Devops	PostgreSQL	Support
Training	Audit	Migrations

https://kamedata.com

Disclaimer

- Non sono un programmatore
- Sono un DBA
- Ovvero sono odiato da tutti ed odio tutti
- Per mettere le cose nella giusta prospettiva...
- Io uso i TABS!

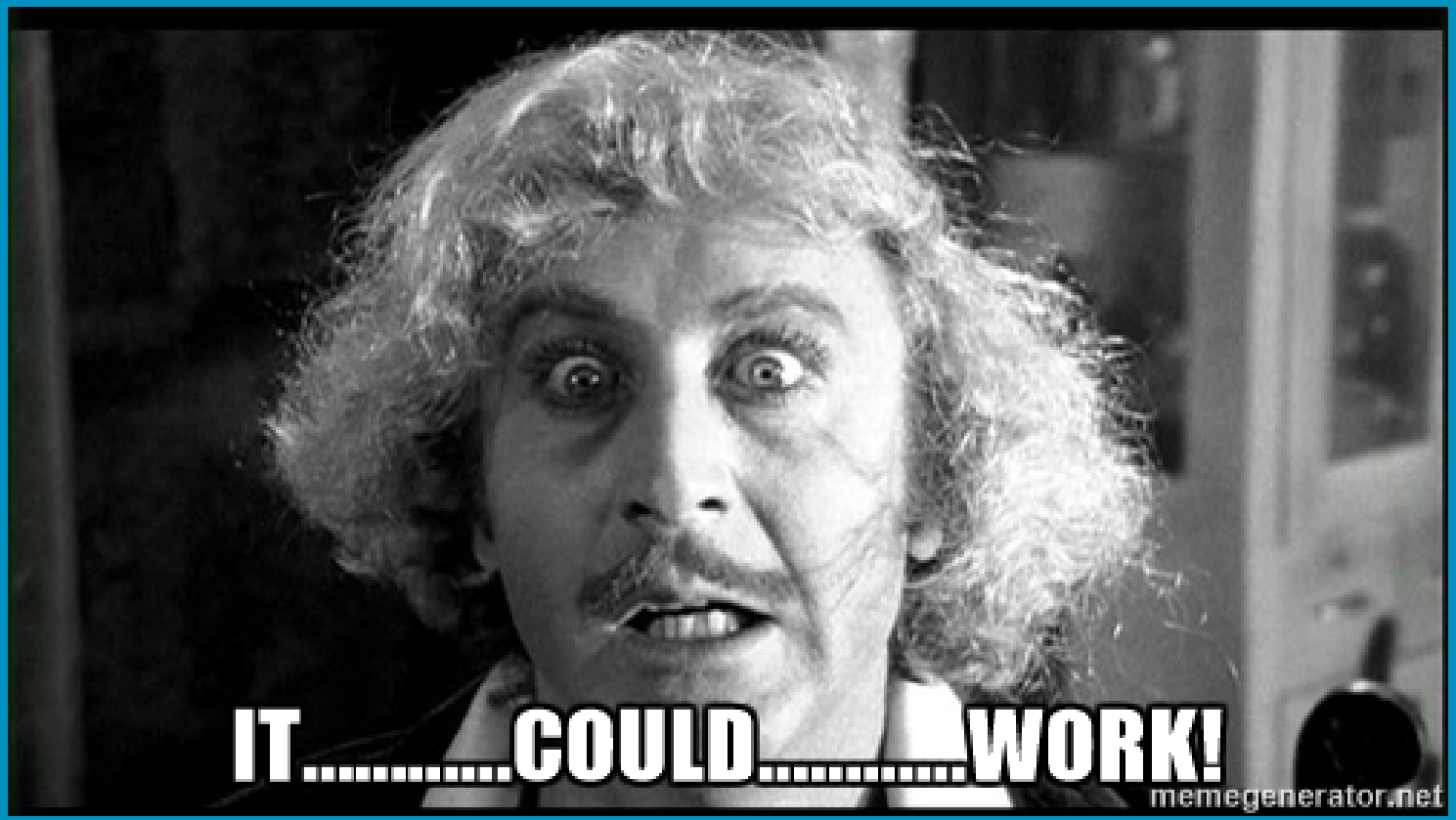
Palpatine

Agenda

KAMEDATA
DEVOPS AND DATA ENGINEERING

1. Introduzione
2. pg_chameleon 2.0
3. la replica in azione
4. Conclusioni

Un po' di storia



Come tutto ebbe inizio

KAMEDATA
DEVOPS AND DATA ENGINEERING

Anni 2006/2012

neo_my2pg.py

- Script di migrazione fatto per scalare un phpbb su MySQL
- Nei miei sogni...
- La migrazione del database andò bene
- Risorse sistema esaurite con PostgreSQL
- phpbb apre una nuova connessione database per ogni query...

Come tutto ebbe inizio

KAMEDATA
DEVOPS AND DATA ENGINEERING

- Lo script è scritto in python 2.6
- È monolitico
- Ed è lento, molto lento
- Può essere usato come checklist per cose da evitare quando si programma
- https://github.com/the4thdoctor/neo_my2pg

Ho un ORM e non ho paura di usarlo

KAMEDATA
DEVOPS AND DATA ENGINEERING

Anni 2013/2015
pg_chameleon tentativo numero uno

- Sviluppato in python 2.7
- SQLAlchemy usato per estrarre i metadati MySQL
- Prototipo molto rudimentale
- Sviluppato durante gli anni del life on a roller coaster
- Life on a roller coaster: https://youtu.be/R86dJcXofZg

Ho un ORM e non ho paura di usarlo

KAMEDATA
DEVOPS AND DATA ENGINEERING

- Per lo più un modo per scaricare la frustrazione
- Progetto abbandonato dopo qualche mese
- Le limitazioni di SQLAlchemy incredibilmente frustranti
- C'era già pgloader che faceva lo stesso lavoro, molto molto meglio

Il ritorno di pg_chameleon

KAMEDATA
DEVOPS AND DATA ENGINEERING

Anno 2016

- Nuovo lavoro, con un problema spinoso. Replicare i dati da MySQL a PostgreSQL
- La storia è descritta qui http://tech.transferwise.com/scaling-our-analytics-database/
- Grazie alla libreria python-mysql-replication riuscii a creare un prototipo.
- Che divenne poi pg_chameleon 1.x
- Non potrò mai ringraziare abbastanza il team di sviluppo della libreria python-mysql-replication!

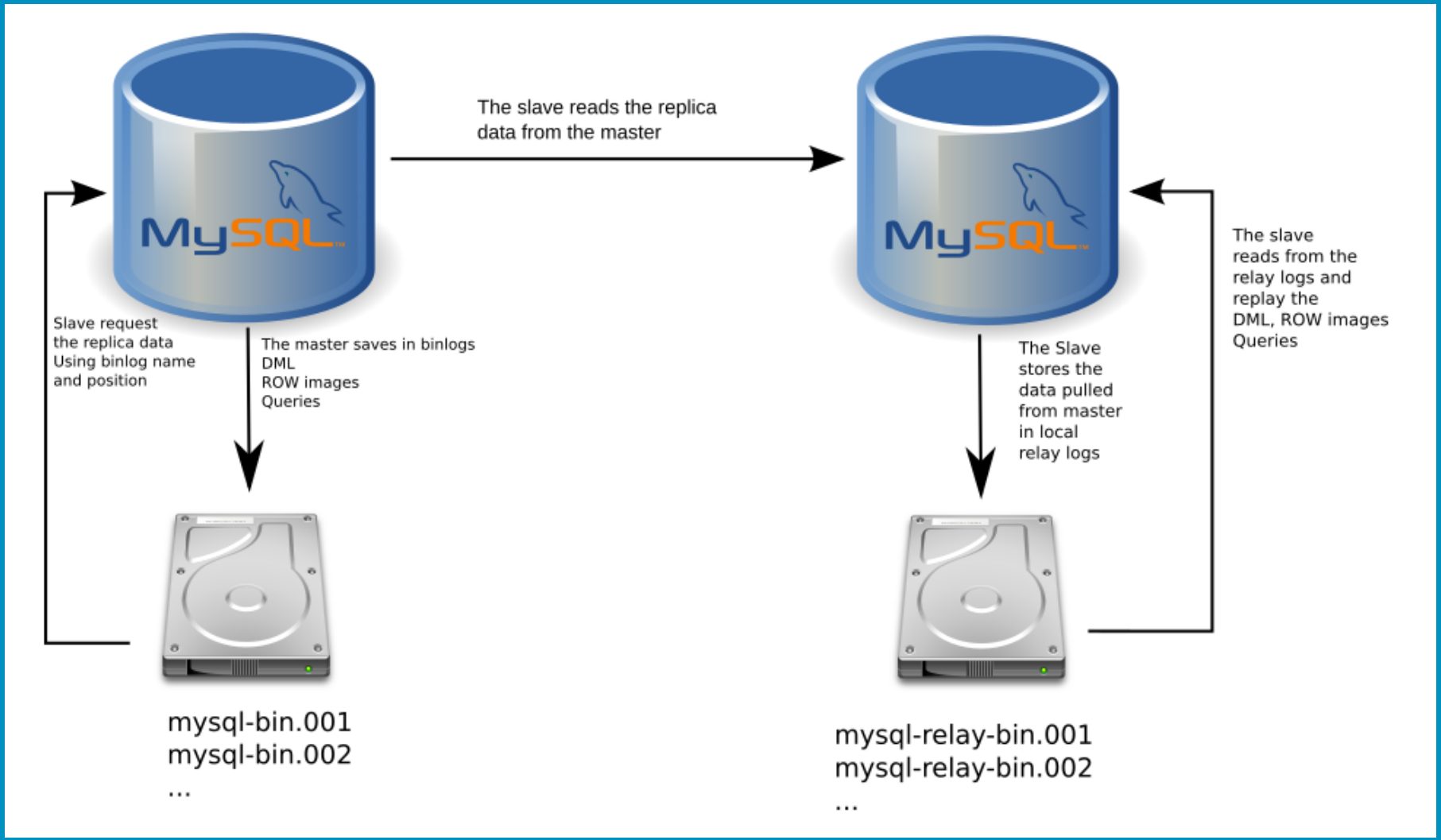
- https://github.com/noplay/python-mysql-replication

La replica in MySQL

Replica MySQL

- La replica in MySQL è logica
- Quando configurato correttamente, MySQL salva le modifiche ai dati nei log binari del master
- Lo slave si collega al master e ne legge i log file binari
- Lo stream di dati è salvato sullo slave in appositi log binari chiamati log di relay
- I dati dei log di relay sono usati per replicare le modifiche nello slave
- In MySQL le modifiche ai dati possono essere loggate in tre modi diversi

Replica MySQL



Formati di log

STATEMENT: Memorizza il comando DML nel log. La replica semplicemente riesegue il comando SQL letto dal binlog.

Pur essendo un formato di log efficiente, presenta il rischio concreto di corruzione dati.

Ciò avviene se vengono usate funzioni non deterministiche nelle DML.

Formati di log

ROW: Questo formato memorizza l'immagine della riga modificata che viene applicata dalla replica localmente.

Formati di log

MIXED: Combina ROW e STATEMENT. Il master logga i comandi DML a meno che una funzione non deterministica sia usata nella DML. In tal caso l'immagine di riga viene loggata al posto del SQL.

Tutti e tre i formati loggano gli eventi DDL.
La libreria python-mysql-replication usata da pg_chameleon richiede il formato ROW per funzionare correttamente.

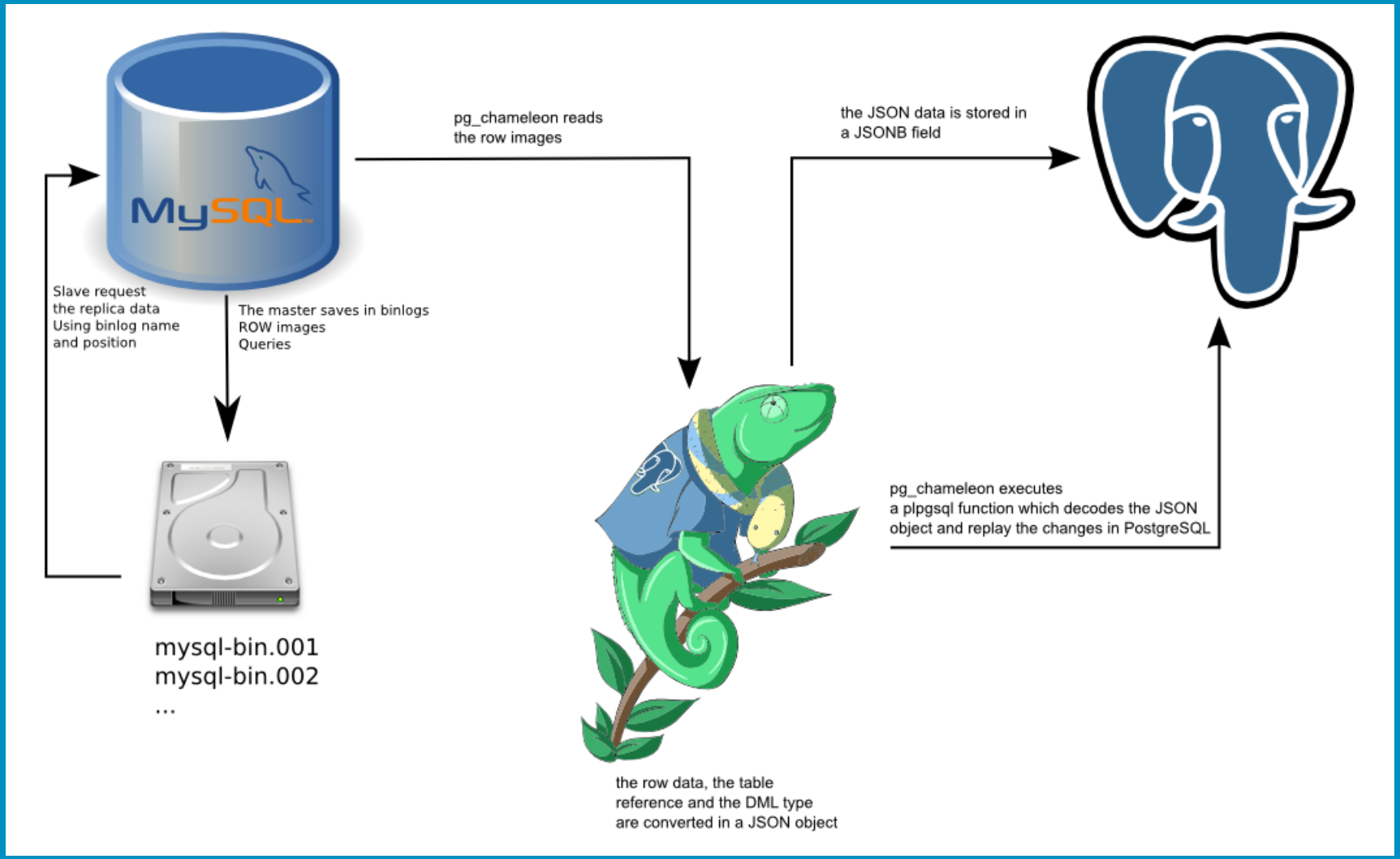
pg_chameleon 2.0

KAMEDATA
DEVOPS AND DATA ENGINEERING

pg_chameleon simula il comportamento di uno slave MySQL con PostgreSQL che agisce sia come relay log che come replica slave

- Esegue il caricamento iniziale dei dati delle tabelle replicate
- Si connette al protocollo di replica di MySQL
- Memorizza le immagini delle righe in una tabella di PostgreSQL
- Una funzione PlpgSQL decodifica le righe ed effettua la replay sul database PostgreSQL
- Può disconnettere la replica per effettuare delle migrazioni di motore con un downtime minimo

Replica MySQL con pg_chameleon



pg_chameleon 2.0 #1

KAMEDATA
DEVOPS AND DATA ENGINEERING

- Sviluppato durante la conferenza pgconf.eu 2017 e sulla tratta Brighton/London
- Rilasciato come stabile il primo gennaio 2018
- Compatibile con python 3.4+
- Installazione in virtualenv o system wide via pypi
- Può replicare schemi multipli dalla stessa sorgente MySQL in un database PostgreSQL

pg_chameleon 2.0 #2

KAMEDATA
DEVOPS AND DATA ENGINEERING

- Approccio alla replica conservativo.
  Le tabelle che generano errori durante la replay vengono escluse dalla replica
- Demonizzazione completa. Un processo per ogni operazione: watchdog, lettura e replay
- Lock tabelle ad impatto ridotto durante l'inizializzazione della replica.
- Possibilità di escludere eventi (INSERT, UPDATE,DELETE) per singole tabelle o interi schemi
- Integrazione con rollbar per semplificare l'alert e il debug di eventuali errori

pg_chameleon 2.0 #3

KAMEDATA
DEVOPS AND DATA ENGINEERING

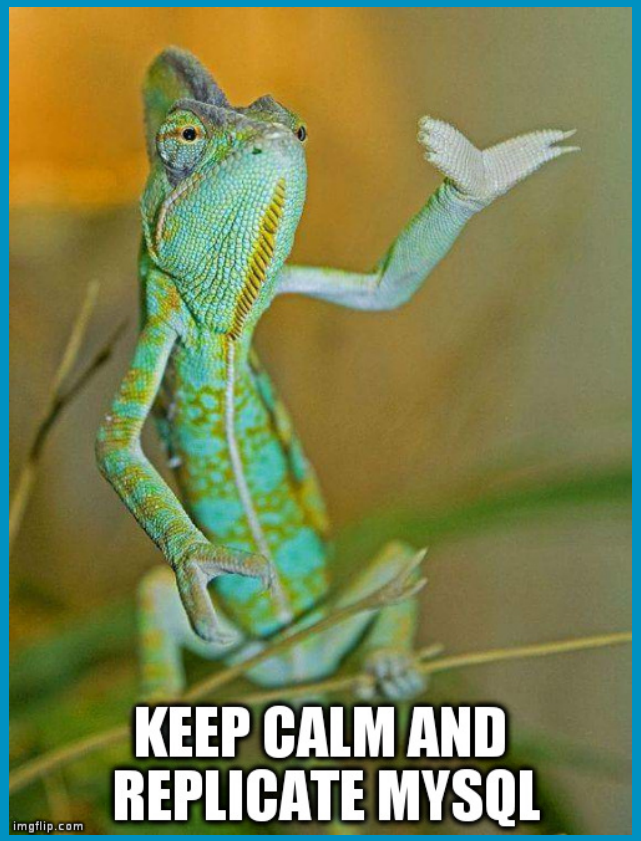
- Supporto sperimentale della sorgente di tipo PostgreSQL
- Durante l'inizializzazione le tabelle sono create e i dati caricati in uno schema separato.
- Le DDL generate su MySQL sono tradotte nel dialetto di PostgreSQL mantenendo gli schemi in sync automaticamente.
- Supporto MySQL GTID. È possibile cambiare la sorgente mysql all'interno del cluster GTID senza dover inizializzare di nuovo la replica.

Limitazioni della versione 2.0

KAMEDATA
DEVOPS AND DATA ENGINEERING

- Le tabelle per essere replicate devono avere una primary o unique key
- Con detach_replica le the foreign keys su PostgreSQL sono sempre create come ON DELETE/UPDATE RESTRICT
- La sorgente tipo PostgreSQL supporta solo il comando init_replica
- L'implementazione GTID di MariaDB non è supportata (limitazione a livello di libreria)

La replica in azione



Inizializzazione della replica

L'inizializzazione della replica segue lo stesso percorso indicato nella documentazione MySQL.

- Flush delle tabelle con il read lock
- Lettura delle coordinate del master
- Copia dei dati
- Rilascio del read lock

Inizializzazione della replica

KAMEDATA
DEVOPS AND DATA ENGINEERING

Tuttavia...

- pg_chameleon effettua il flush con read lock una tabella alla volta
- La tabella è tenuta in sola lettura solo durante la copia
- Le coordinate in cui il lock è avvenuto vengono salvate per la tabella nel catalogo di replica in PostgreSQL
- Il processo di lettura usa le coordinate per determinare quando iniziare a usare le immagini del log binario

In caso di errore usare la insert

Durante init_replica i dati sono estratti da MySQL in formato CSV (comma separated values) e in slice multiple. Questo approccio permette di controllare il quantitativo di memoria usata durante la copia.

Quando il file con i dati è salvato questi viene caricato in PostgreSQL usando il comando COPY.

In caso di errore usare la insert

KAMEDATA
DEVOPS AND DATA ENGINEERING

Tuttavia...

- COPY è veloce ma è in singola transazione
- Un errore e l'intero comando viene annullato con un rollback
- Se questo avviene la procedura tenta di caricare gli stessi dati usando delle INSERT, riga per riga
- In caso di errore pg_chameleon tenta di ripulire i marcatori NUL dalla riga che danno problemi con PostgreSQL
- Se la insert fallisce di nuovo allora la riga viene saltata

Consigurazione di MySQL

KAMEDATA
DEVOPS AND DATA ENGINEERING

Il file di configurazione di MySQL, su Linux si trova normalmente in /etc/mysql/my.cnf
Per abilitare il logging binario bisogna trovare la sezione [mysqld] e verificare che i seguenti parametri siano settati.

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

Utente MySQL per la replica

KAMEDATA
DEVOPS AND DATA ENGINEERING

In seguito configurare un utente con i permessi necessari per la 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;
```

Per questa dimostrazione useremo il database di esempio sakila
https://dev.mysql.com/doc/sakila/en/

Setup di PostgreSQL

KAMEDATA
DEVOPS AND DATA ENGINEERING

In PostgreSQL è sufficiente creare un utente senza particolari permessi ed un database il cui owner sia questo utente.

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

Installare pg_chameleon

Installare pg_chameleon e creare i file di configurazione di esempio.

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

Copiare il file di esempio in default.yml e modificarlo con i valori corretti per le connessioni sorgente e target.

Configurazione dei settaggi globali in default.yml

KAMEDATA
DEVOPS AND DATA ENGINEERING

Connessione PostgreSQL

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

Configurazione dei settaggi globali in default.yml

KAMEDATA
DEVOPS AND DATA ENGINEERING

Configurazione rollbar

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

Override di tipo (opzionale)

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

Configurazione sorgente MySQL

KAMEDATA
DEVOPS AND DATA ENGINEERING

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

```
    schema_mappings:
      sakila: loxodonta_africana
```

Configurazione sorgente MySQL

KAMEDATA
DEVOPS AND DATA ENGINEERING

```
    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'
```

Configurazione sorgente MySQL

KAMEDATA
DEVOPS AND DATA ENGINEERING

```
    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
```

Configurazione sorgente MySQL

KAMEDATA
DEVOPS AND DATA ENGINEERING

```
    skip_events:
      insert:
        - sakila.author #skips inserts on the table sakila.author
      delete:
        - sakila #skips deletes on schema sakila
      update:
```

Inizializzazione replica

KAMEDATA
DEVOPS AND DATA ENGINEERING

Creare il catalogo di replica, aggiungere la sorgente ed inizializzare la replica. Per maggiore chiarezza usiamo l'opzione –debug.

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

Avviare la replica

KAMEDATA
DEVOPS AND DATA ENGINEERING

Avviare il processo di replica

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

Visualizzare lo stato di replica

```
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
--------------------  -----
```

Ed ora una demo!

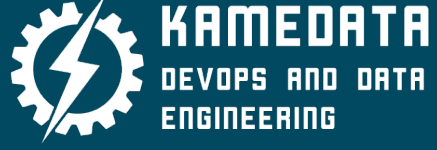
KAMEDATA
DEVOPS AND DATA ENGINEERING

Demo!

Ovviamente, la demo fallirà miseramente e voi odierete questo progetto per sempre.

Conclusioni

Strictness is an illusion. MySQL doubly so



La maniera in cui MySQL gestisce i default non espliciti con NOT NULL possono interrompere il processo di replica.

Per evitare ciò ogni campo aggiunto alle tabelle dopo l'inizializzazione della replica, sia esso NULL o NOT NULL in PostgreSQL viene creato sempre senza constraint sul NULL.

Convertire le DDL, un lavoro di pazienza

KAMEDATA
DEVOPS AND DATA ENGINEERING

Replicare le DDL è fondamentale per mantenere sincronizzata la struttura delle tabelle, tra MySQL e PostgreSQL. Sfortunatamente i dialetti tra i due motori sono diversi.
Per tokenizzare DDL generate da MySQL, tentai inizialmente di usare la libreria sqlparse. Purtroppo la libreria si dimostrò fragile, difficile da usare e con funzionalità incomplete.

Pertanto decisi di usare le regular expression.

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

-- Jamie Zawinski
```

Convertire le DDL, un lavoro di pazienza

KAMEDATA
DEVOPS AND DATA ENGINEERING

- MySQL anche se configurato con il formato ROW logga le query delle DDL eseguite
- La classe sql_token usando le regular expression trasforma in token le DDL
- Il token viene poi usato per ricostruire la DDL nel dialetto PostgreSQL

Dove nessun camaleonte è mai giunto prima!

KAMEDATA
DEVOPS AND DATA ENGINEERING

Versione 2.1

- Lo sviluppo della versione 2.1 è già iniziato
- Il rilascio era pianificato per inizio 2019....
- Ma Murphy ci mette sempre lo zampino...

Dove nessun camaleonte è mai giunto prima!

KAMEDATA
DEVOPS AND DATA ENGINEERING

Migliorie rispetto alla versione 2.0

- Copia dati e creazione indici in parallelo per migliorare la velocità del processo init_replica
- Supporto localizzazione (in italiano per ora)
- Replica da altri RDBMS, si inizia con PostgreSQL
- Migliorare la gestione dei default NULL
- Sync automatico delle tabelle in caso di esclusione dalla replica

Igor, il camaleonte verde

Il logo di pg_chameleon è stato realizzato dalla talentuosa Elena Toma.

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

Il nome Igor è ispirato dal personaggio Igor interpretato da Martin Feldman in Frankenstein Junior.

Feedback per favore!

KAMEDATA
DEVOPS AND DATA ENGINEERING

Per favore inviate bug report su github e seguite pg_chameleon su twitter per gli annunci.

https://github.com/the4thdoctor/pg_chameleon
@pg_chameleon

That's all folks!

Domande?

KAMEDATA
DEVOPS AND DATA ENGINEERING

Per favore niente di complicato, alla fine sono solo un elettricista.

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
- Framed - Copyright Federico Campoli
- Pinkie Pie that's all folks, Copyright by dan232323, used with permission

License

pg_chameleon

Replicare MySQL in PostgreSQL con semplicità

Federico Campoli

Kamedata

PGDay Italia, Bologna 17 Maggio 2019