Percona Live Europe 2016

ProxySQL Tutorial

High Performance & High Availability
Proxy for MySQL

With a GPL license!

Amsterdam, Netherlands | October 3 – 5, 2016

Visit us at booth #102

Who we are

David Turner

DBA, Uber

Derek Downey

OSDB Practice Advocate, Pythian

René Cannaò

MySQL SRE, Dropbox / ProxySQL

Main motivations

empower the DBAs

improve operation

understand and improve performance

create a proxy layer to shield the database

High performance and High Availability

ProxySQL Features

Some of the most interesting features:

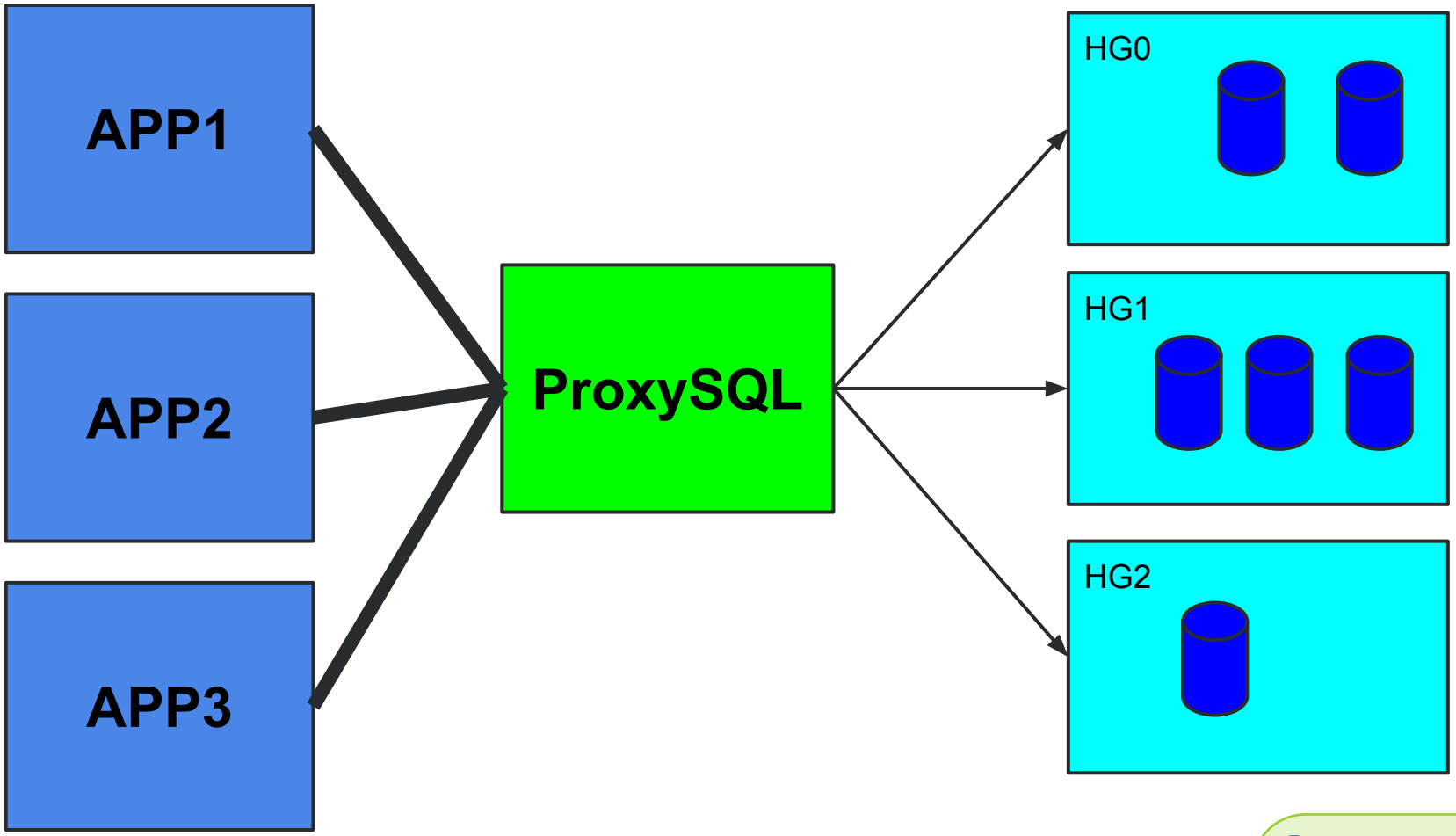
1. on-the-fly rewrite of queries
2. caching reads outside the database server
3. connection pooling and multiplexing
4. complex query routing and read/write split
5. load balancing
6. real time statistics
7. monitoring
8. High Availability and Scalability
9. seamless failover
10. firewall
11. query throttling
12. query timeout
13. query mirroring
14. runtime reconfiguration

Hostgroups and Query Routing

All backends are grouped into hostgroups

Hostgroups have logical functionalities

Basic design



Hostgroups example #1

HostGroup0 (HG0): Write masters

HostGroup1(HG1): Read slaves

Read/Write split

Hostgroups example #2

HG0: main write masters

HG1: main read slaves

HG2: reporting slaves

HG3: ad-hoc queries slaves

HG4: data warehouse write masters

HG5: data warehouse read slaves

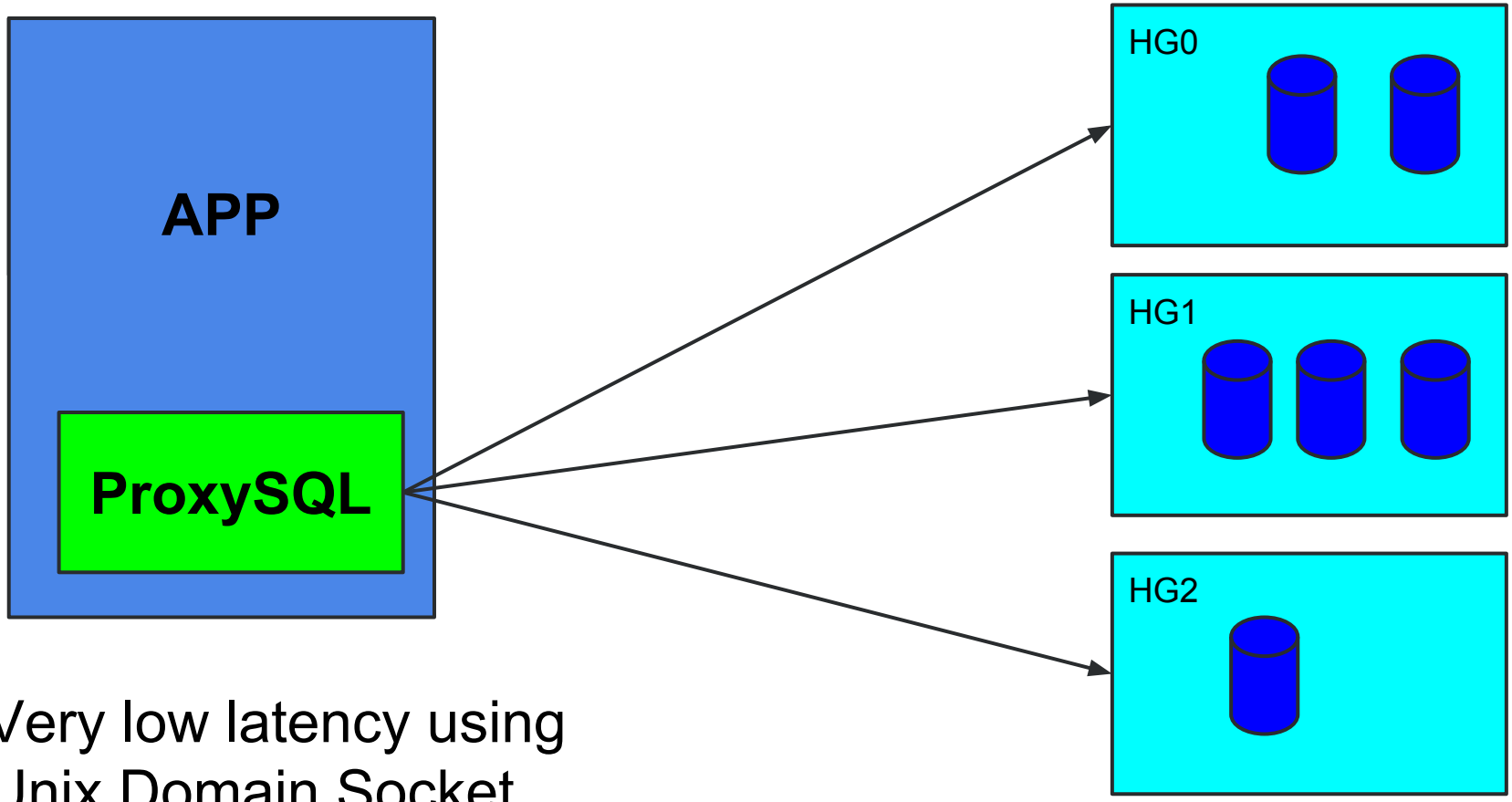
HG6: remote site servers

HG7: test servers

HG8 : mirror for traffic on HG0

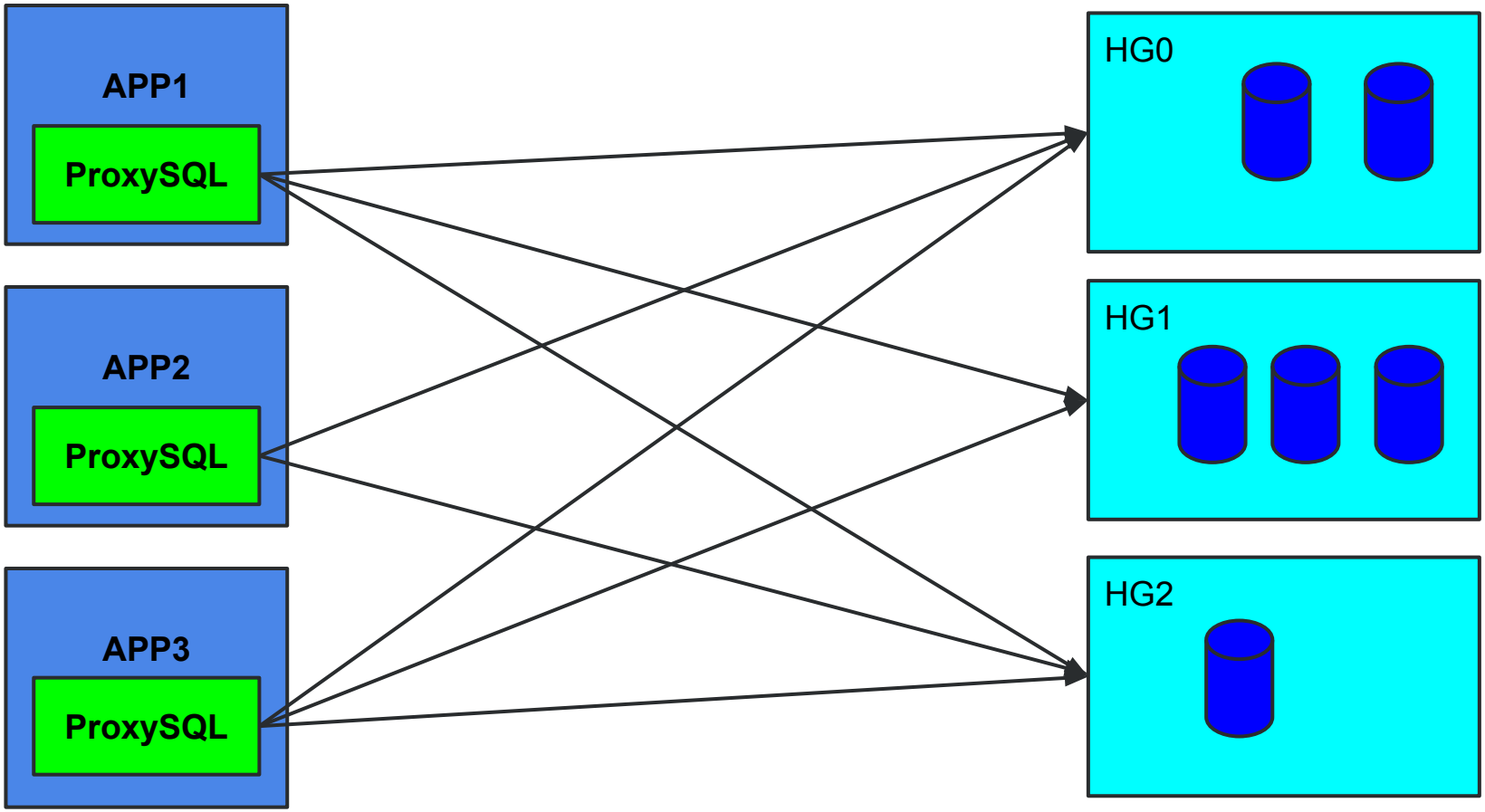
HG9 : mirror for traffic on HG1

Basic design

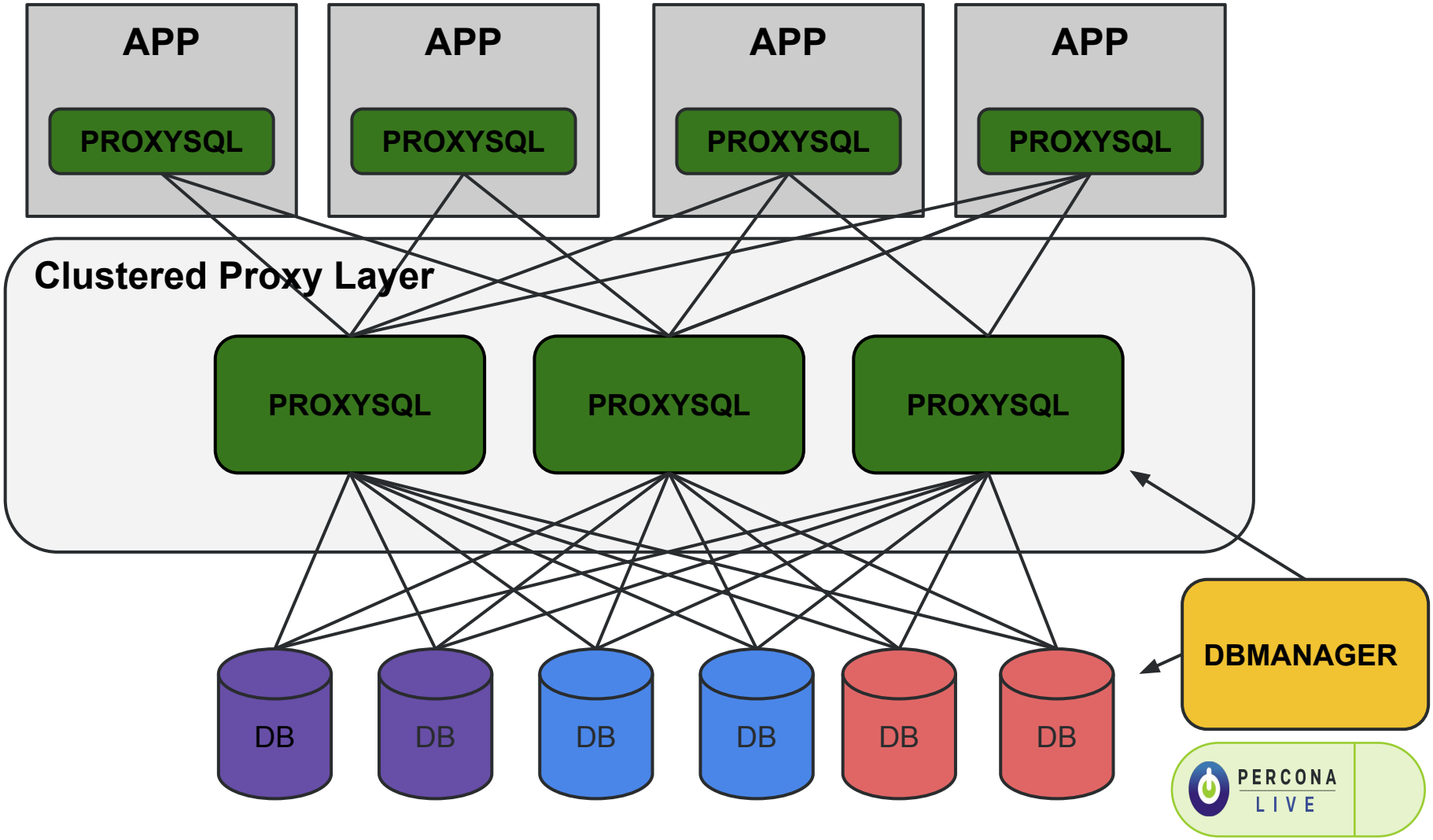


Very low latency using
Unix Domain Socket

Basic design



Clustered ProxySQL Architecture



Clustered ProxySQL at scale

Tested with:

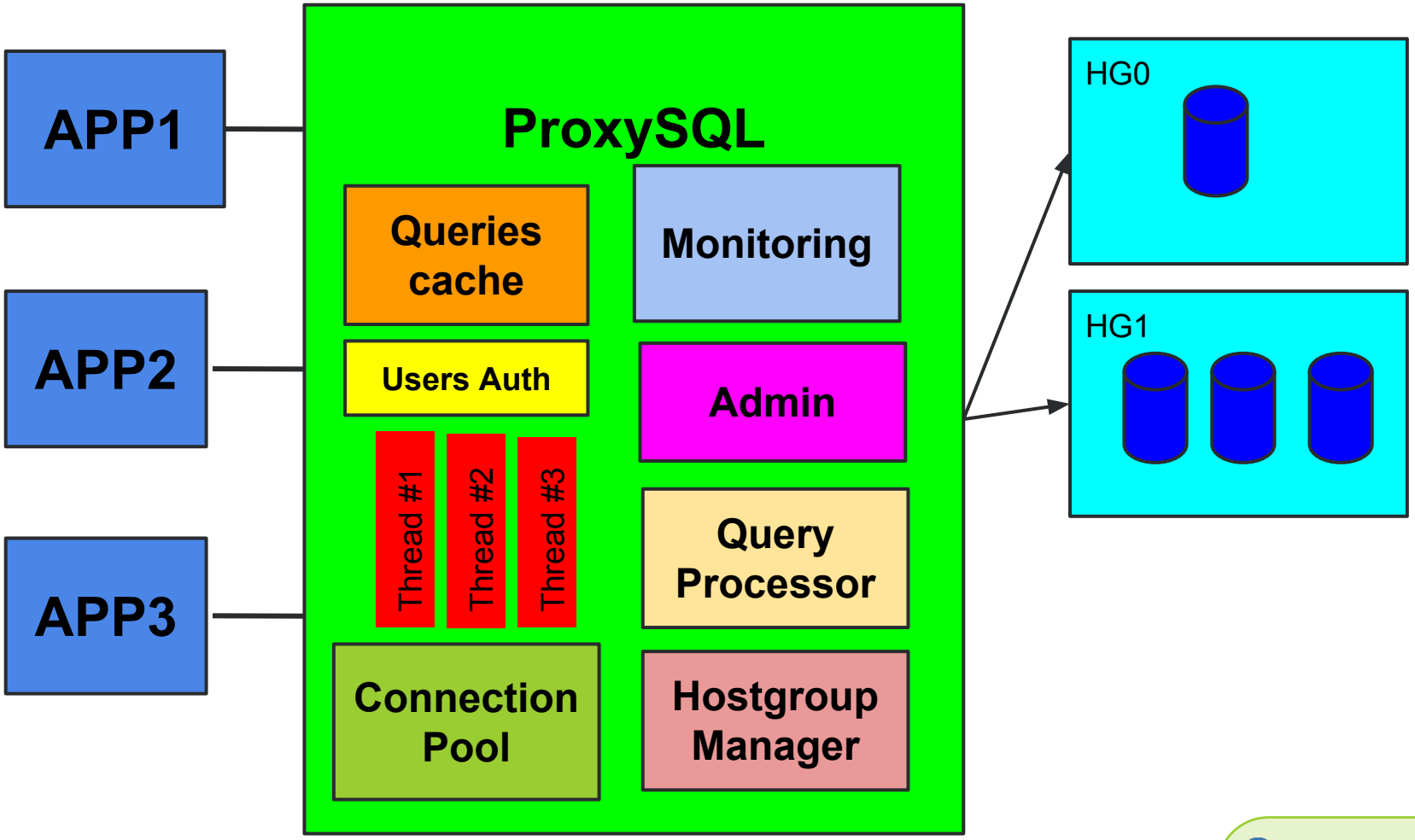
- 8 app servers with 3k clients' connections each (24k total)
- 4 middle layer proxysqls processing 4k connections each from local proxysqls (16k total)
- 256 backends/shard (meaning 256 routing rules) processing 600 connections each (150k total)

Single ProxySQL was tested with up to 150k connections

At today, ProxySQL is able to process up to 750k QPS

ProxySQL Internals

ProxySQL Modules



Queries Processor

Based on Queries Rules

Defines what to cache

Defines the hostgroup target

Timeout/delay

Firewall

Mirroring

Rewrite queries

Queries rules

Complex rules to match incoming traffic:

- regex on query
- regex on digest text
- username
- schemaname
- Source IP address
- Bind IP address/port
- digest

Rules can be chained

Queries Cache and Rewrite

Caching on the wire

Internal key/value storage

In memory only

Pattern based

Expired by timeout

Rewrite on the wire

Regex match/replace on query on digest text

Optionally cached or mirrored

Users Authentication

Credentials stored in the proxy

User login always possible (even without backends)

Max connections

Login credentials are encrypted

Hostgroups Manager

Management of servers

Track servers status

Tightly integrated with the connections pool

Connections Pool

Reduced the overhead of creating new connections, and are recycled when not in use

One to many connections

Multiplexing & maximum connections

Auto-reconnect and automatic re-execution of queries

Failover management

Auto-reconnect and re-execution

Automatic detection of failures

Graceful handling

Auto-reconnect when possible

Pause until a backend becomes available

Re-execution of queries

Multiplexing

Reduce the number of connections against mysqld (configurable)

Many clients connections (tens of thousands) can use few backend connections (few hundreds)

Tracks connection status (transactions, user variables, temporary tables, etc)

Order by waiting time

Monitoring Module

It monitors backends and collects metrics

Monitors replication lag and shun hosts

Monitors read_only variables (replication hostgroups)

Ping and terminates unresponsive nodes

Failover with ProxySQL

Failover

2 phases process:

- remove host
- add host

Seamless switchover:

http://proxysql.blogspot.com/2015/09/proxysql-tutorial-seamless-replication.html

Managed by external process

Switchover in less than 1 second

Distributed failover

Multiple ProxySQL are available in a network

Failover managed by an external process that:

- remove host from each ProxySQL instance
- add new host into each ProxySQL instance

Manager is not part of ProxySQL.

Ex: MHA or MySQL Utilities

Replication hostgroups

Constantly monitor read_only variable

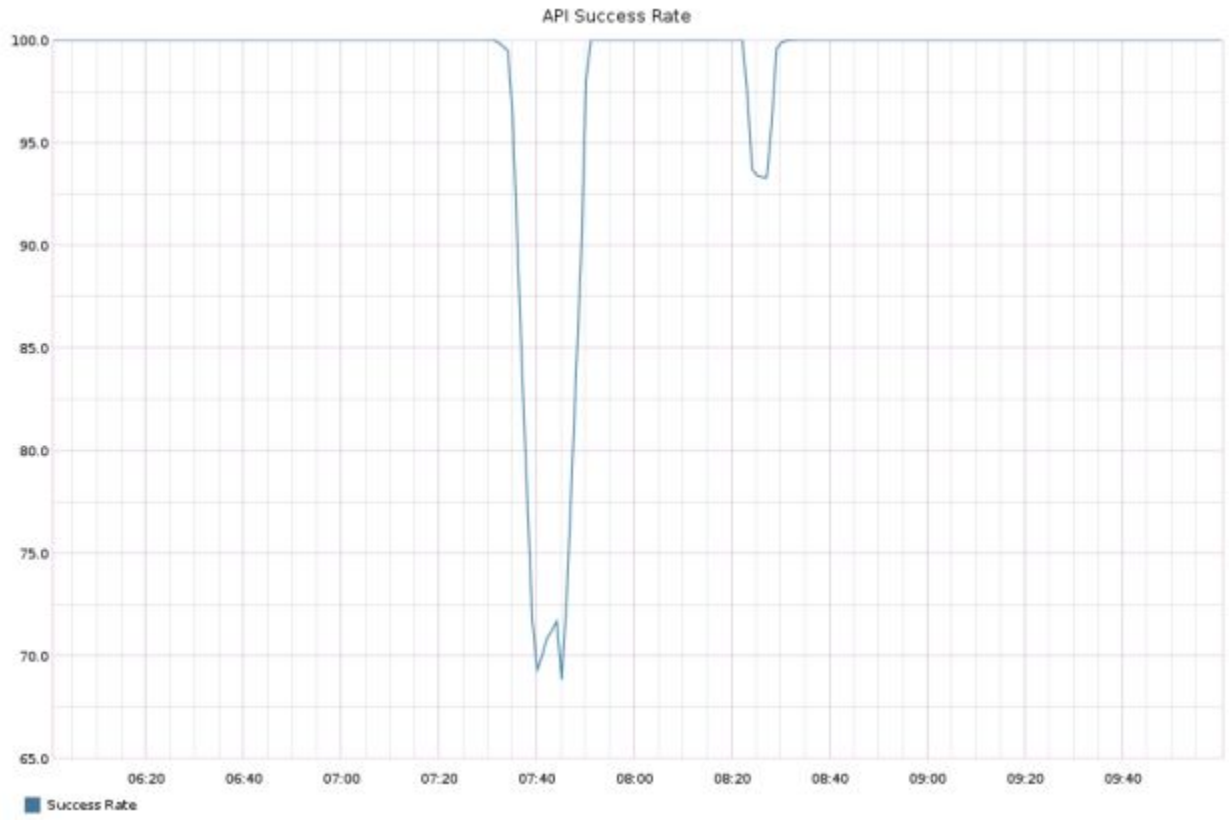
Defines replication topology as writer(s)/readers(s)

Automatically re-assign servers to the right hostgroup

Possible to define unlimited number of replication clusters

Replication hostgroups

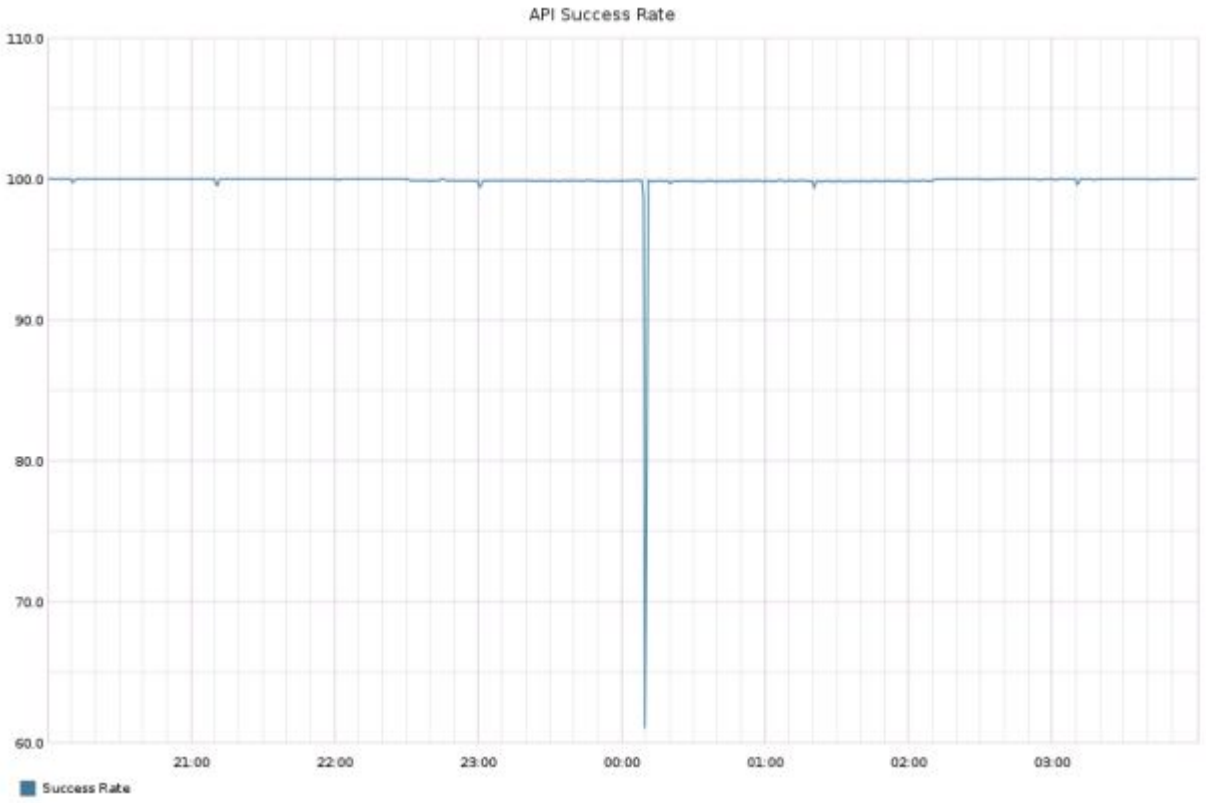
Failover without ProxySQL (10 minute outage)



https://www.nylas.com/blog/growing-up-with-mysql/

Replication hostgroups

Failover with ProxySQL (10 second outage)



https://www.nylas.com/blog/growing-up-with-mysql/

Failover highlight

improve failover time as perceived by the application

prevent errors sent to the application

perform transparent database failovers: gracefully redirecting traffic without the application knowing

existing applications do not have to be rewritten to autoreconnect since connections are not lost from failovers

Admin Module

Admin Interface

Allows runtime configuration

Exports internal statuses

It uses MySQL protocol

Configuration possible from any client/tool using MySQL API

Covered during the tutorial

Try it!

Source code on GitHub:
https://github.com/sysown/proxysql/

Forum:
https://groups.google.com/forum/#!forum/proxysql

Tutorials on:
http://www.proxysql.com

Join us at booth #102

Demo environment

Demo

Code on GitHub:

https://github.com/dtest/plam16-proxysql

Run locally: Docker, Ansible

Some hosts provided:

Username: plam

Passwords: proxysql

The 3 Rs of ProxySQL and related statistics

ProxySQL

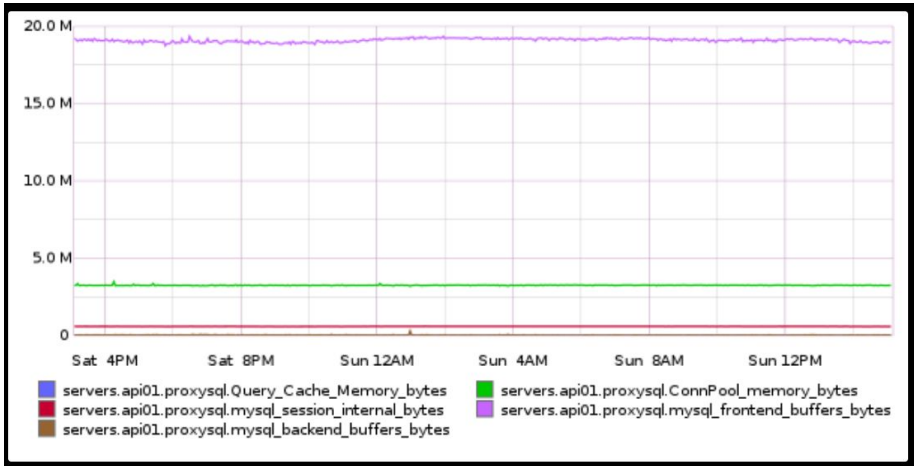
- The 3 Rs
  - Rules
  - Rewrite
  - Routing
- Related Statistics
  - Identify offensive queries and resources consumed
  - View results of Remediation

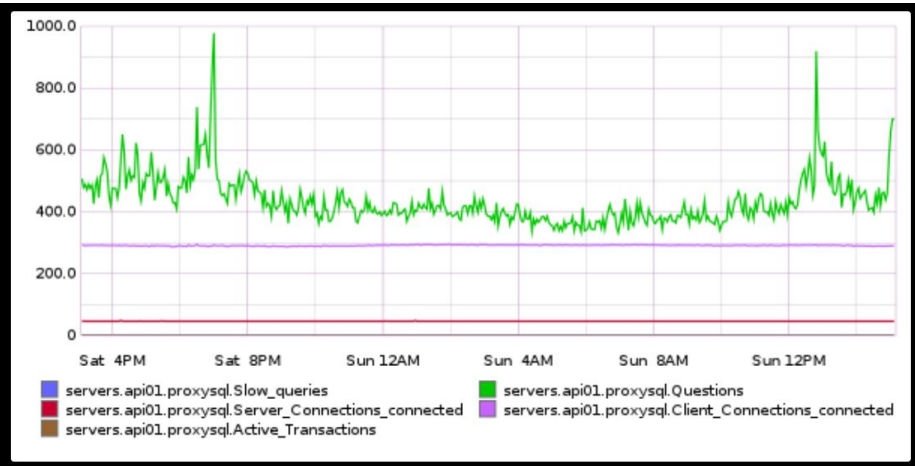
ProxySQL Troubleshooting Methodology

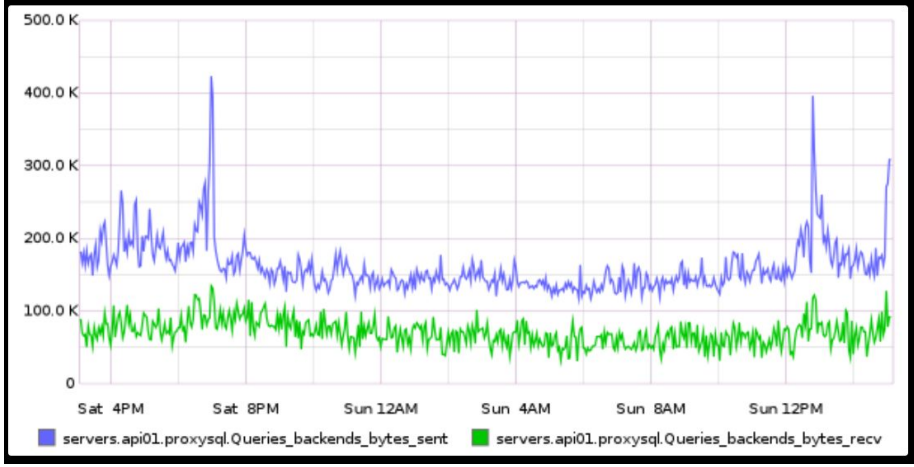
- Identify problem via stats tables
- Arrive at a solution
  - Match columns
  - Action columns

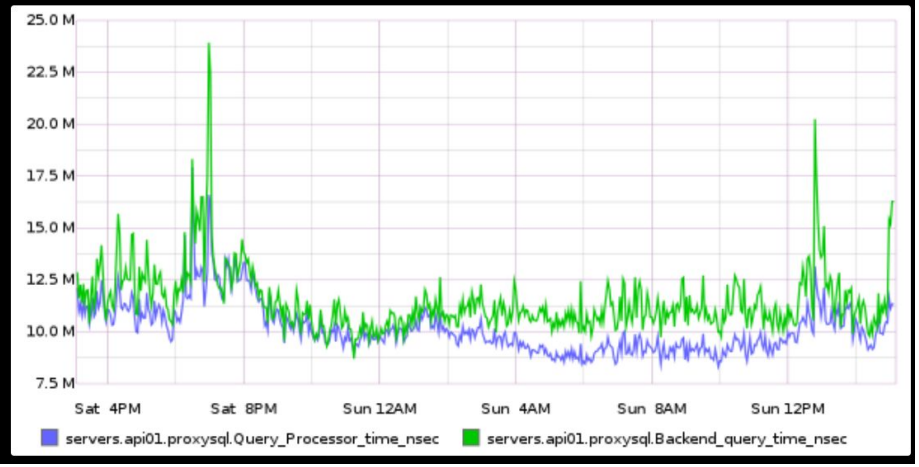
Identify the problem

- stats_mysql_query_rules
- stats_mysql_commands_counters
- stats_mysql_processlist
- stats_mysql_connection_pool
- stats_mysql_query_digest
- stats_mysql_query_digest_reset
- stats_mysql_global


20.0 M
15.0 M
10.0 M
5.0 M
0
Sat 4PM
Sat 8PM
Sun 12AM
Sun 4AM
Sun 8AM
Sun 12PM
servers.api01.proxysql.Query_Cache_Memory_bytes
servers.api01.proxysql.ConnPool_memory_bytes
servers.api01.proxysql.mysql_session_internal_bytes
servers.api01.proxysql.mysql_frontend_buffers_bytes
servers.api01.proxysql.mysql_backend_buffers_bytes


1000.0
800.0
600.0
400.0
200.0
0
Sat 4PM
Sat 8PM
Sun 12AM
Sun 4AM
Sun 8AM
Sun 12PM
servers.api01.proxysql.Slow_queries
servers.api01.proxysql.Questions
servers.api01.proxysql.Server_Connections_connected
servers.api01.proxysql.Client_Connections_connected
servers.api01.proxysql.Active_Transactions


500.0 K
400.0 K
300.0 K
200.0 K
100.0 K
0
Sat 4PM
Sat 8PM
Sun 12AM
Sun 4AM
Sun 8AM
Sun 12PM
servers.api01.proxysql.Queries_backends_bytes_sent
servers.api01.proxysql.Queries_backends_bytes_recv


25.0 M
22.5 M
20.0 M
17.5 M
15.0 M
12.5 M
10.0 M
7.5 M
Sat 4PM
Sat 8PM
Sun 12AM
Sun 4AM
Sun 8AM
Sun 12PM
servers.api01.proxysql.Query_Processor_time_nsec
servers.api01.proxysql.Backend_query_time_nsec

Match Related Tables

- mysql_rules
  - username
  - schemaname
  - flagIN
  - client_addr
  - proxy_addr
  - proxy_port
  - digest
  - match_digest
  - match_pattern
  - Negate_match_pattern
- mysql_users
  - active
  - fast_forward(bypass)
- global_variables
- mysql_collations
  - Default(utf8)
  - No stats

Action related fields

- flagOUT
- replace_pattern
- destination_hostgroup
- cache_ttl
- reconnect
- timeout
- retries
- delay
- mirror_flagOUT
- mirror_hostgroup
- error_msg
- log
- apply
- active

Let the fun begin

- ssh <your-ip>
- sudo su -
- cd /root/dba/admin/dev/plam16-proxysql/
- ./run_proxy.sh
- docker exec -it proxysql bash (at least to terminals)
- export TERM=ansi
- cd /root/plam-rrr
- Please resist viewing the files(I will tell you why)

Stats scripts

- show_stats_mysql_commands_counters.sh
- show_stats_mysql_connection_pool.py
- show_stats_mysql_global.sh
- show_stats_mysql_processlist.sh
- show_stats_mysql_query_digest.sh
- show_stats_mysql_query_digest_reset.sh

3 Levels for ProxySQL Tables

- Disk
- Memory
- Runtime

Scenarios

- p1.py
- p2.py
- p3.py
- p4.py

Monitoring scripts

- show_mysql_query_rules.sh
- show_mysql_servers.sh
- show_mysql_users.sh
- show_stats_mysql_commands_counters.sh
- show_stats_mysql_connection_pool.py
- show_stats_mysql_global.sh
- show_stats_mysql_processlist.sh
- show_stats_mysql_query_digest.sh
- show_stats_mysql_query_digest_reset.sh
- show_stats_mysql_query_rules.sh

Precautions

- Confirm the query hits with stats_mysql_query_rules
- double check rules and rewrite results
- hold off on writing rules to disk until you're sure they are working as expected
- have a rollback plan
  - load from disk, then memory to runtime
  - load from repo, then memory to runtime and later load to disk
- test select query rules and rewrites on a slaves first
- test mutable rules on a mirror or other throw away db.

Thank you

- Derek Downey
- Krzysztof Ksiazek
- Marco Tusa

- Rene Cannao

Links

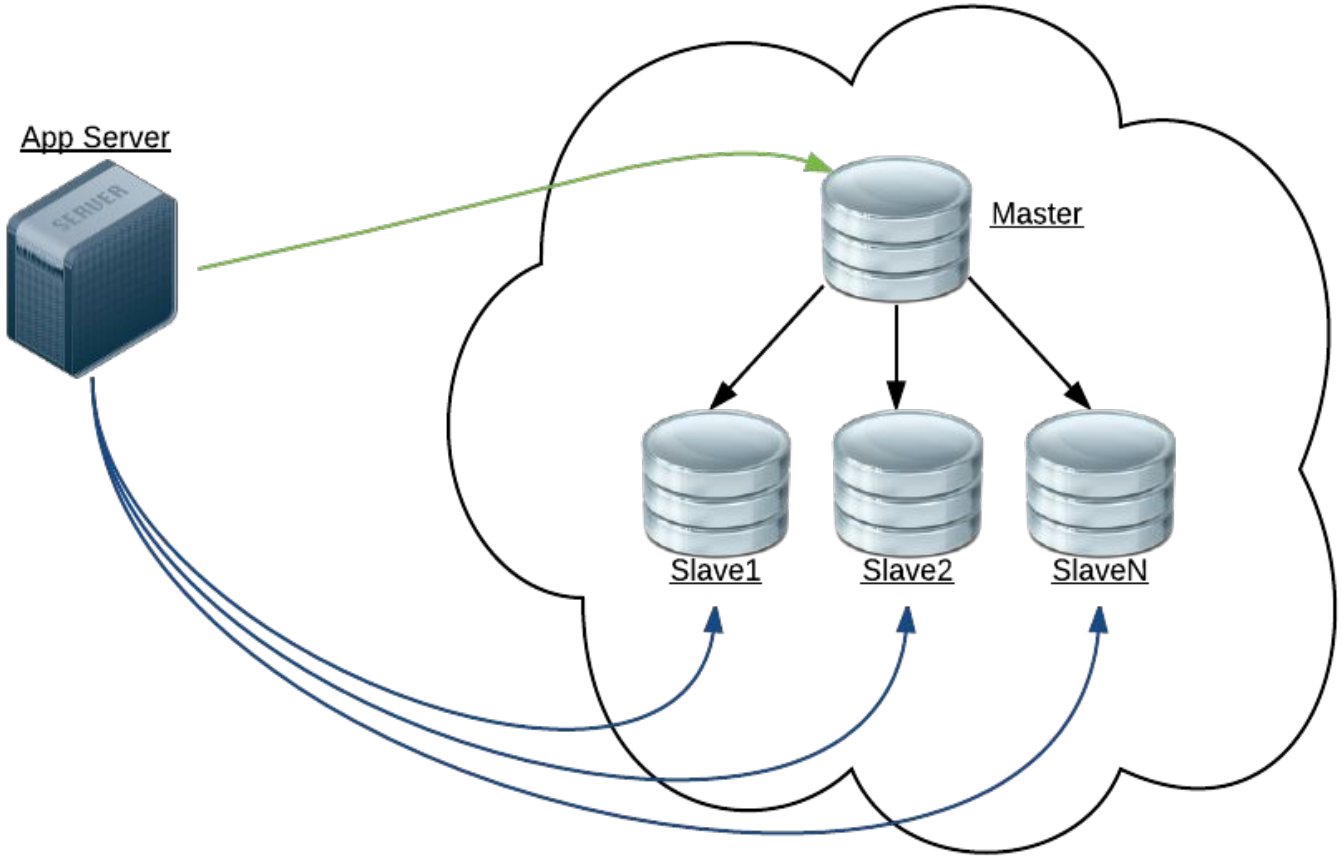
- https://github.com/sysown/proxysql/tree/master/doc
- https://www.percona.com/blog/2016/08/30/mysql-sharding-with-proxysql/
- https://tusacentral.net/joomla/index.php/mysql-blogs/183-proxysql-percona-cluster-galera-integration.html
- http://severalnines.com/blog/how-proxysql-adds-failover-and-query-control-your-mysql-replication-setup

Uber is hiring

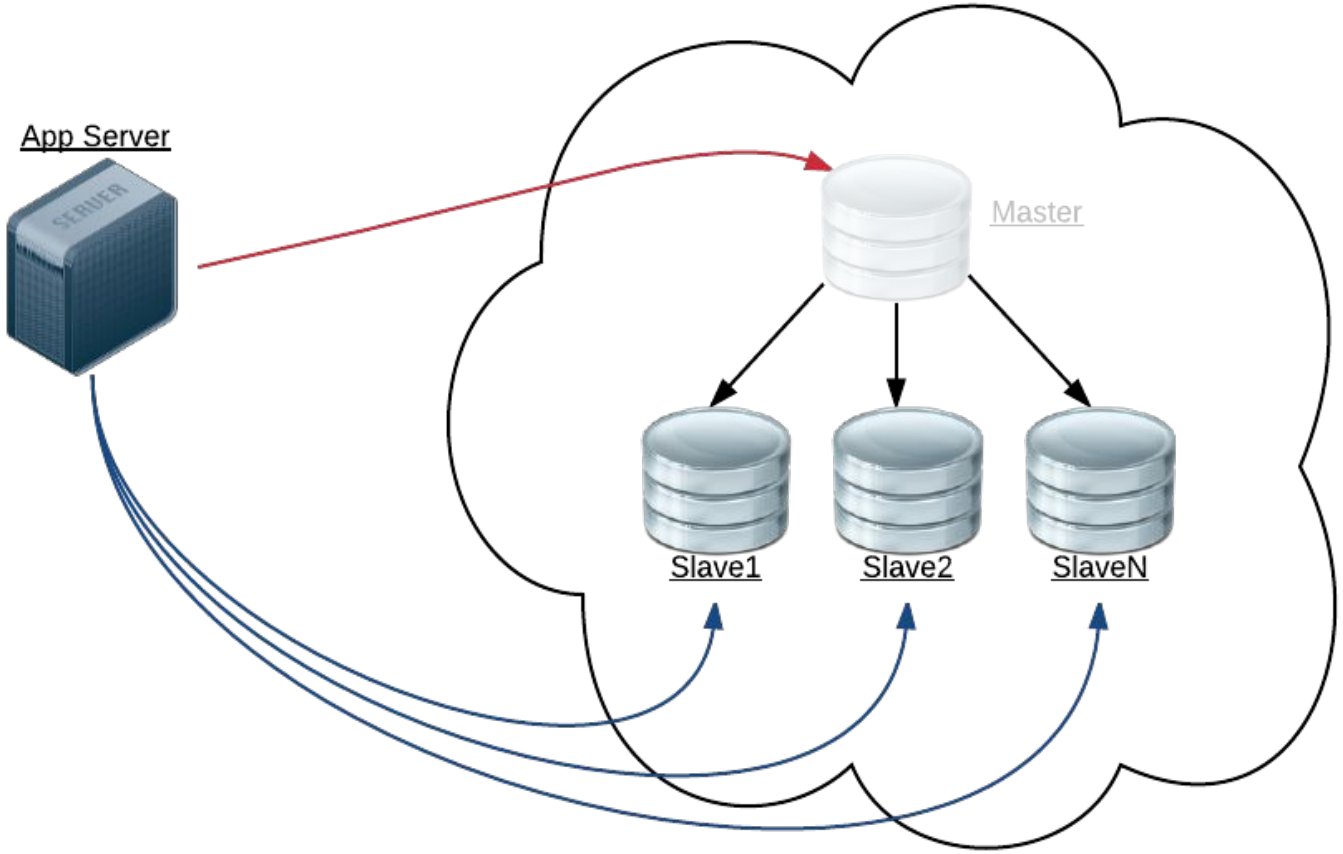
- Growing and talented team
- Embracing Open Source Technologies
- Awesome projects awaiting you
- Cool campuses
- Build the future
- Uber Eats / Monthly credits (weekend coverage)
- Automated vehicles for all employees (Derek)

Failover with ProxySQL

Failover scenario - without ProxySQL



Failover scenario - without ProxySQL



Failover highlights

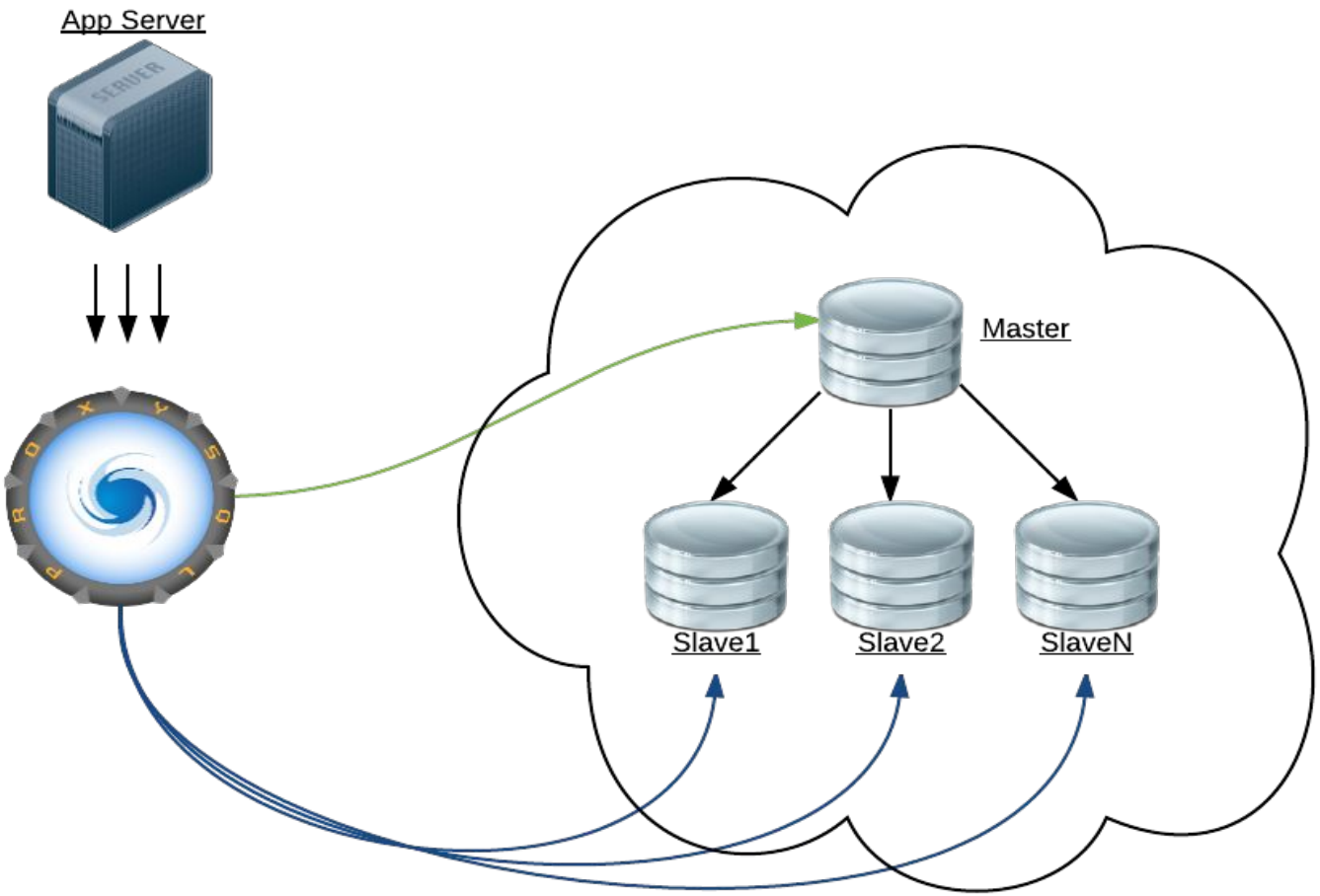
improve failover time as perceived by the application

prevent errors sent to the application

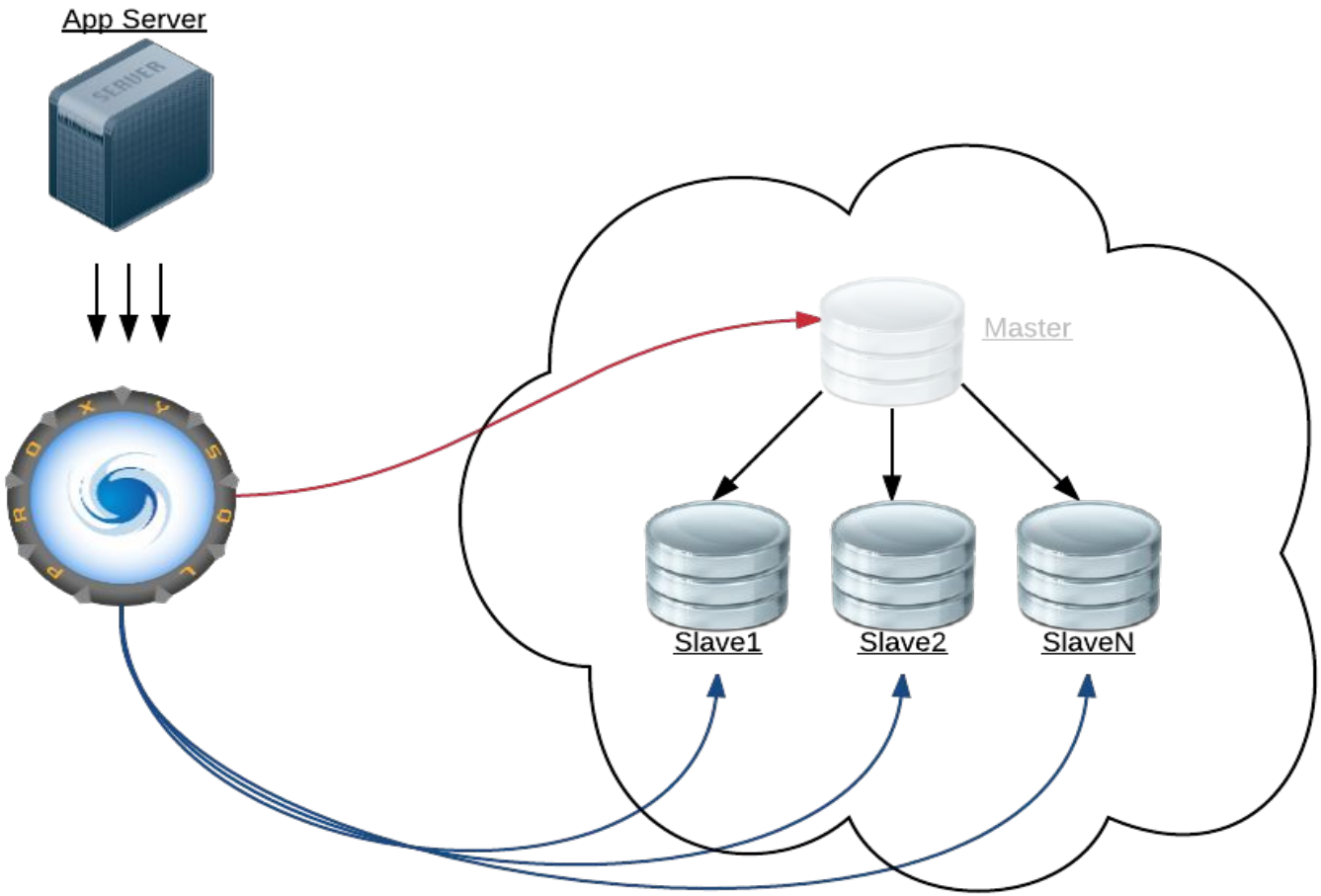
perform transparent database failovers: gracefully redirecting traffic without the application knowing

existing applications do not have to be rewritten to autoreconnect since connections are not lost from failovers

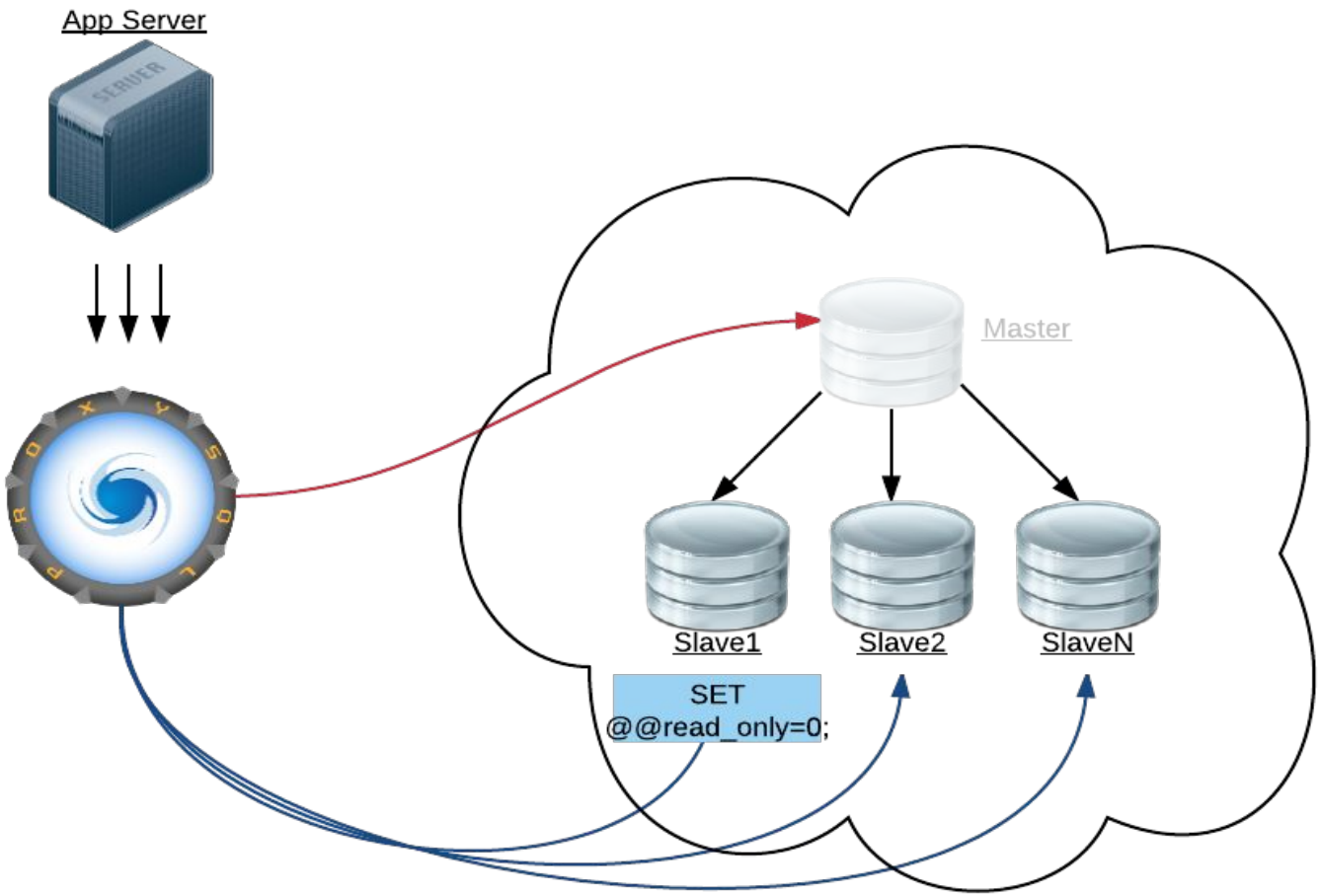
Failover scenario - with ProxySQL



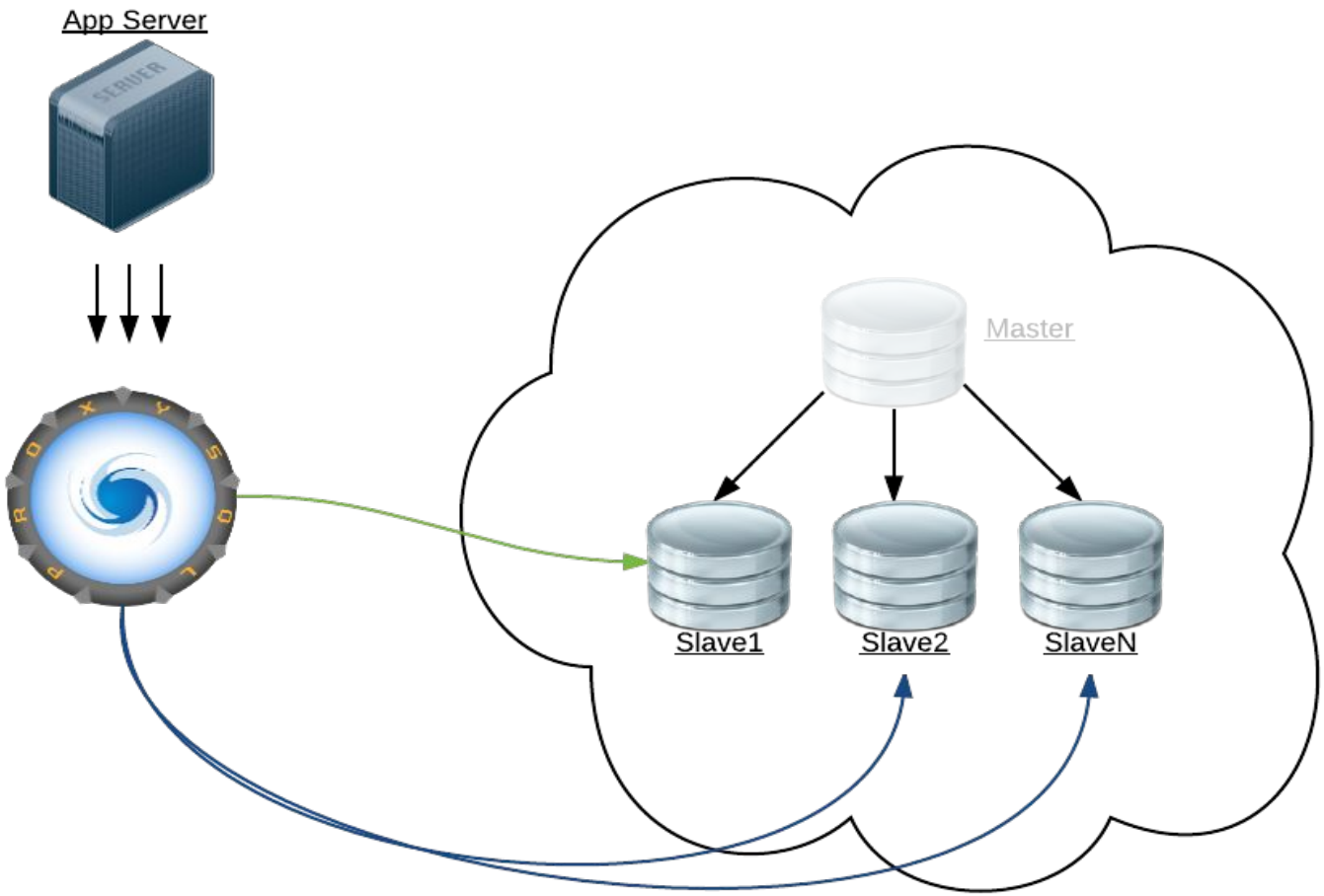
Failover scenario - with ProxySQL



Failover scenario - with ProxySQL



Failover scenario - with ProxySQL



ProxySQL Failover ProTIP

ProxySQL does not handle promotion or re-slaving

External process needed, such as MHA

Failover - MySQL Servers

```
mysql> SELECT hostgroup_id, hostname, status FROM mysql_servers
WHERE hostname IN ('master', 'slave')\G
*************************** 1. row ***************************
       hostgroup_id: 1
       hostname: master
       status: ONLINE
*************************** 2. row ***************************
       hostgroup_id: 2
       hostname: slave
       status: ONLINE
2 rows in set (0.00 sec)
```

Failover - Replication Hostgroups

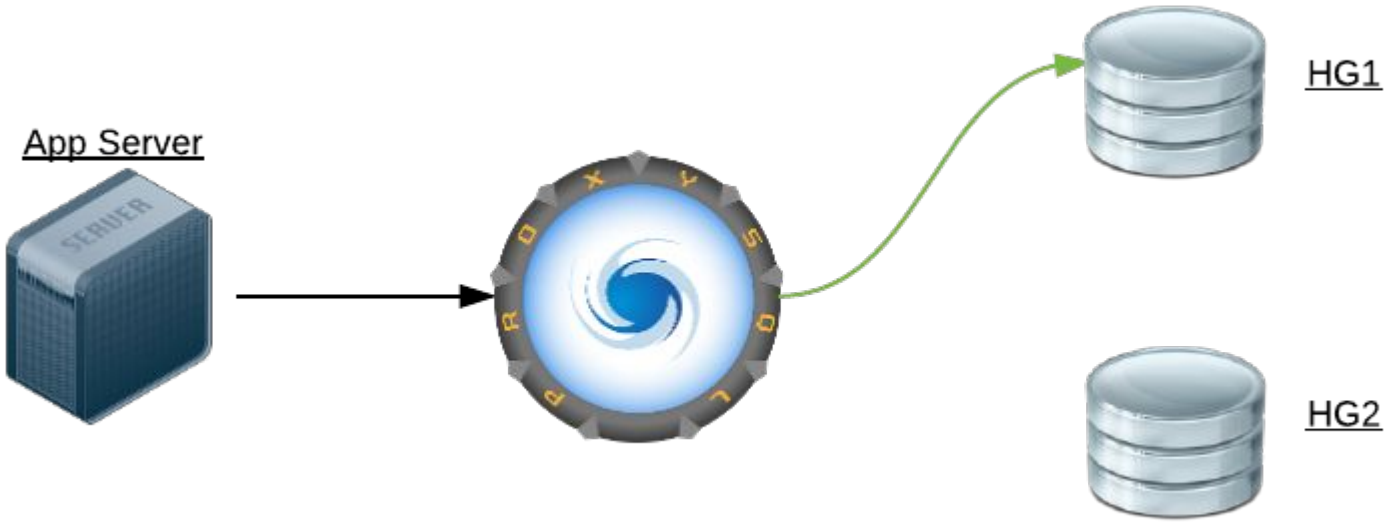
```
mysql> SELECT * FROM mysql_replication_hostgroups\G
*************************** 1. row ***************************
writer_hostgroup: 1
reader_hostgroup: 2
        comment:
1 row in set (0.00 sec)
```

Failover Demonstration

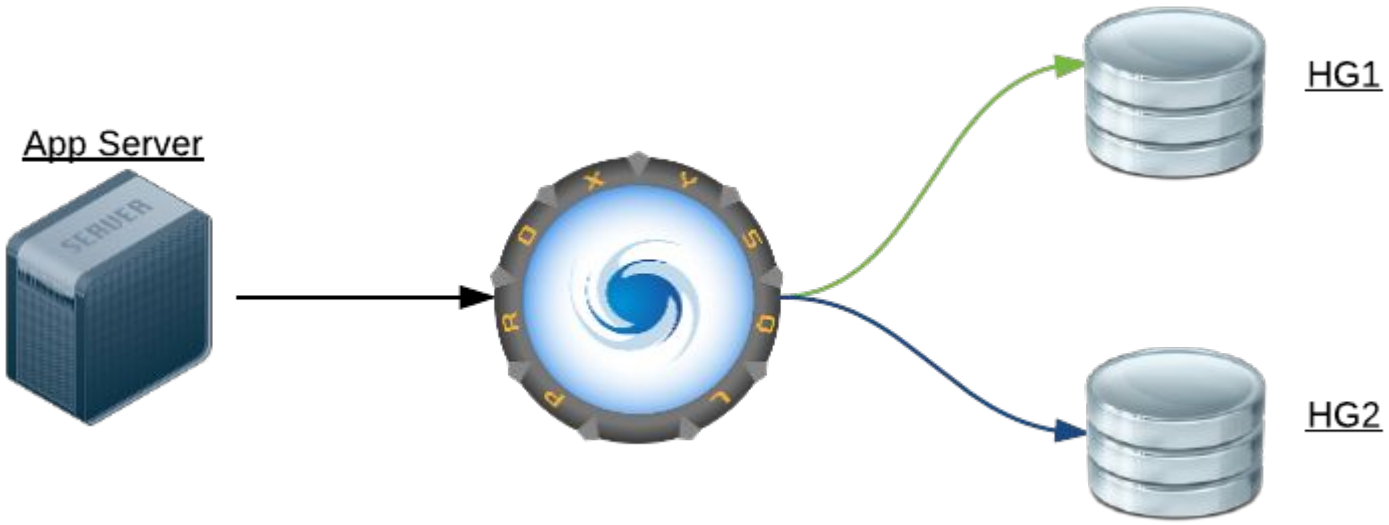
Mirroring with ProxySQL

PERCONA LIVE

What is mirroring?



What is mirroring?



Why mirror queries?

Validate performance on a different server using different hostgroups.

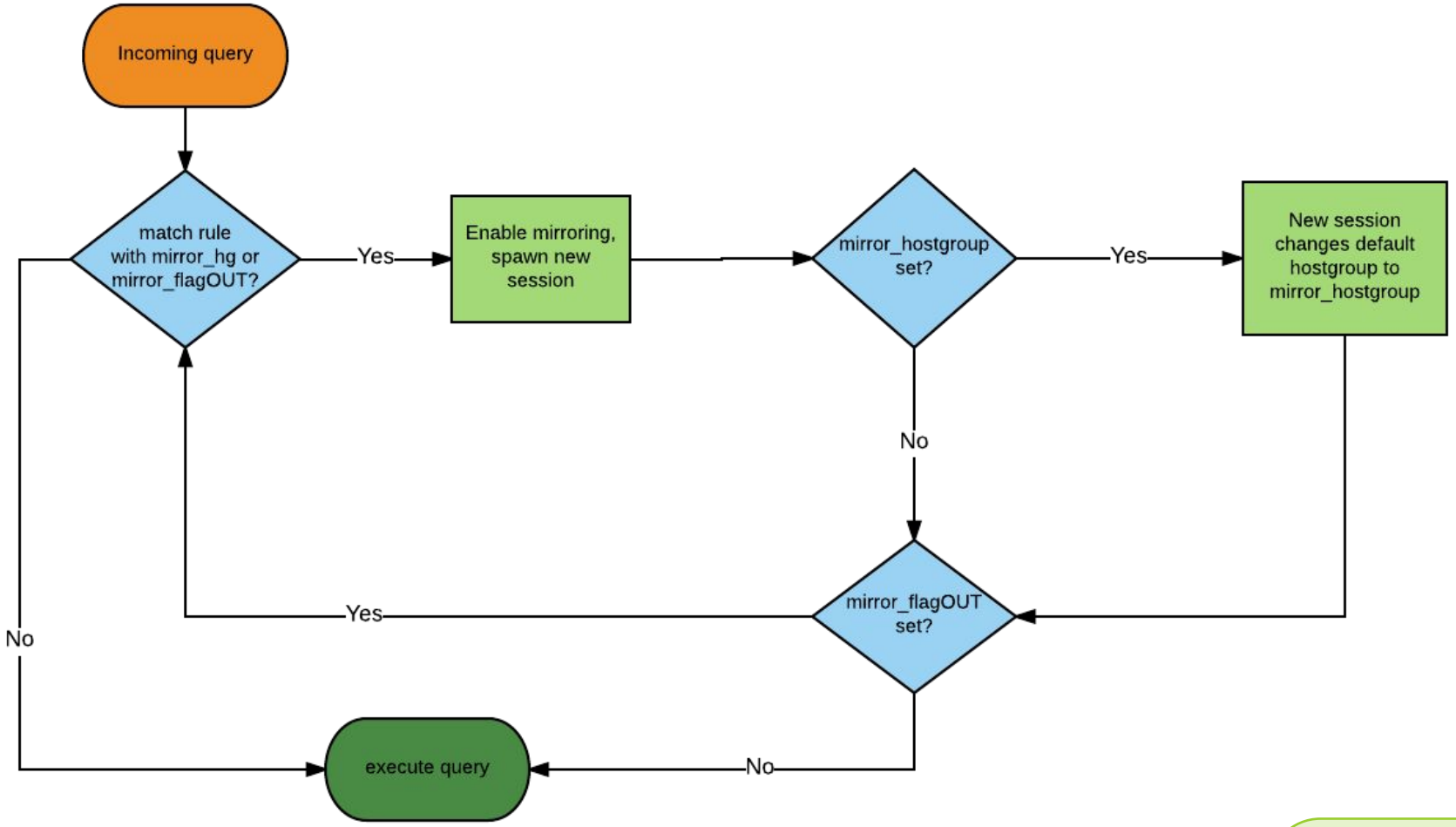
Validate performance of query rewrite or schema change

Pre-fetch slave replication (Replication Booster)

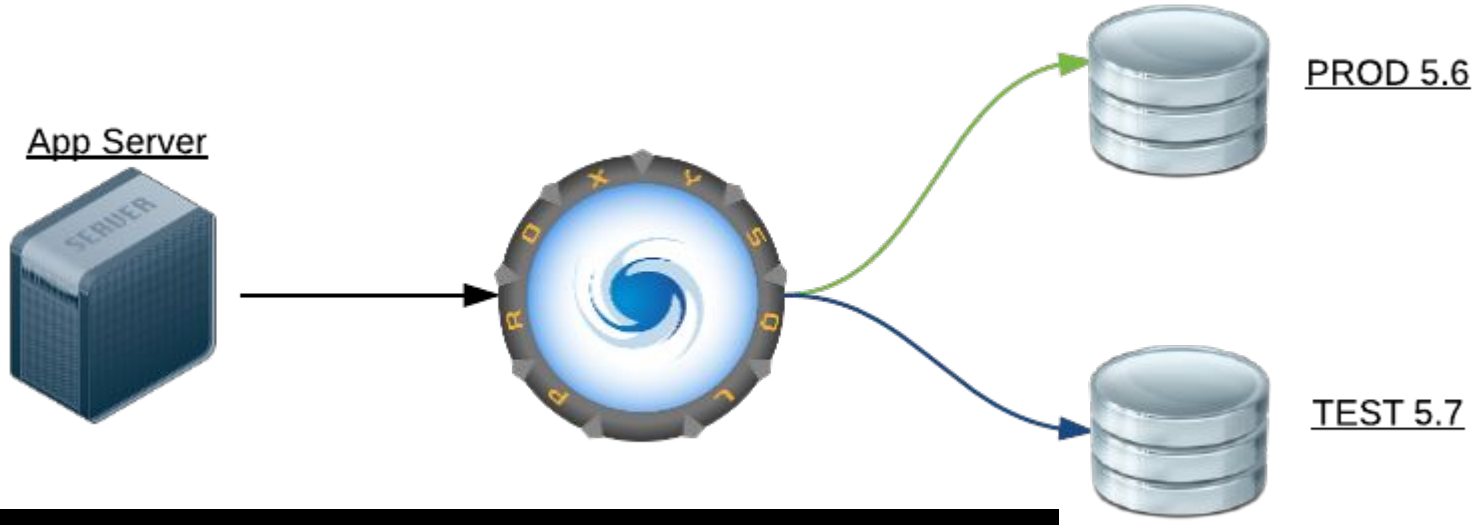
How to mirror

```
mysql>SHOW CREATE TABLE mysql_query_rules\G
*************************** 1. row ***************************
       table: mysql_query_rules
Create Table: CREATE TABLE mysql_query_rules (
       rule_id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
       active INT CHECK (active IN (0,1)) NOT NULL DEFAULT 0,
       username VARCHAR,
       schemaname VARCHAR,
       flagIN INT NOT NULL DEFAULT 0,
*snip*
       mirror_flagOUT INT UNSIGNED,
       mirror_hostgroup INT UNSIGNED,
       error_msg VARCHAR,
       log INT CHECK (log IN (0,1)),
       apply INT CHECK(apply IN (0,1)) NOT NULL DEFAULT 0,
       comment VARCHAR)
```

Mirroring flow

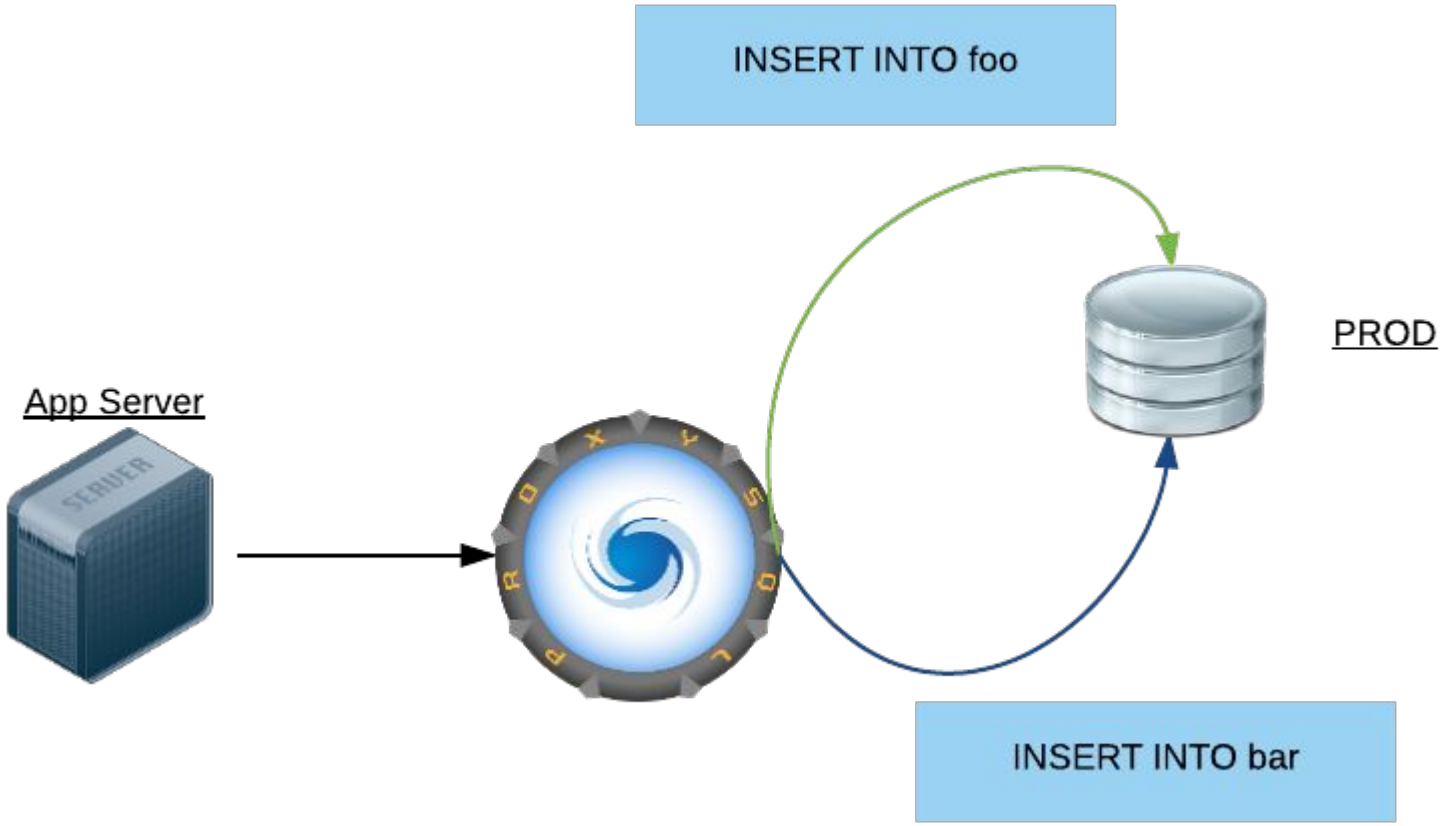


Mirroring - example #1




```
mysql> SELECT username, destination_hostgroup,
mirror_hostgroup, mirror_flagOUT FROM mysql_query_rules
WHERE username='plam_mirror'\G
******************** 1. row ************************
            username: plam_mirror
destination_hostgroup: 3
      mirror_hostgroup: 4
      mirror_flagOUT: NULL
```

Mirroring - example #2



Mirroring Demonstration

Questions?