MySQL Infrastructure Testing Automation @ GitHub

Jonah Berquist, Tom Krouper
GitHub

Percona Live 2018
Agenda

- Intros
- MySQL @ GitHub
- Backup/restores
- Schema migrations
- Failovers
About Tom

• Sr. Infrastructure Engineer
• Member of the Database Infrastructure Team
• Working with MySQL since 2003 (MySQL 4.0 release era)
• Worked on MySQL at Twitter, Booking, and Box previous to GitHub. Several other places too.

https://github.com/tomkrouper
https://twitter.com/@CaptainEyesight
About Jonah

- Infrastructure Engineering Manager
- Member of the Database Infrastructure team
- Proud manager of 5 lovely team members

https://github.com/jonahberquist
https://twitter.com/@hashtagjonah
GitHub

- The world’s largest Octocat t-shirt and stickers store
- And plush Octocats
- And hoodies
- And software development platform
MySQL at GitHub

- GitHub stores repositories in git, and uses MySQL as the backend database for all related metadata.
- We run a few (growing number of) clusters, totaling over 100 MySQL servers.
- The setup isn’t very large but very busy.
MySQL at GitHub

- Our MySQL servers must be available, responsive and in good state
- GitHub has 99.95% SLA
- Availability issues must be handled quickly, as automatically as possible.
Your data

It’s important
Backups

• xtrabackup
• On busy clusters, dedicated backup servers.
• Backups from replicas in each DC
• We monitor for number of “success” events in past 24-ish hours, per cluster.
Restores

- Something bad happened and you need that data
- Building a new host
- Rebuilding a broken one
- All the time!
Restores - the old way

- Dedicated restore servers.
- One per cluster.
- Continuously restores, catches up with replication, restores, catches up with replication, restores, …
- Sending a “success” event at the end of each cycle.
- We monitor for number of “success” events in past 24-ish hours, per cluster.
auto-restore replicas

production replicas

master

backup replica

auto-restore replica

How people build software
Restores - the new way

• Database-class servers in kubernetes.
• Data not persistent.
• Database cluster agnostic.
• Continuously restores, catches up with replication, restores, catches up with replication, restores, ...
• Sending a “success” event at the end of each cycle.
• We monitor for number of “success” events in past 24-ish hours, per cluster.
auto-restore replicas on k8s
Picks a backup from cluster A
starts replicating from cluster A
replication catches up
moves on to backup of cluster B
replicates from cluster B
replication catches up
auto-restore replica not always running
Restores

- New host provisioning uses same flow as restore.
- A human may kick a restore/reclone manually.
  - This can grab the latest, or really any backup we have
- We can also restore from another running host.
restore failure

- A specific backup/restore may fail because of computers.
- No reason for panic:
  - Previous backup/Restores proven to be working
  - At most we lose time
- Lack of successful restore for a cluster in the last ~24 hours is an issue to be investigated
Restore: delayed replica

- One delayed replica per cluster
- Lagging at 4 hours
Backup/restore: logical

- We routinely run a logical backup of all individual tables (independently)
- We can load a specific table from a specific logical backup, onto a non-production server
- No need for DBA. Table allocated in a developer’s space.
- Operation is audited.
Schema migrations
Is your data correct?

The data you see is merely a ghost of your original data.
gh-ost

- Young. 1yr old.
- In production at GitHub since born.
- Software
- Bugs
- Development
- Bugs
gh-ost

- Overview
Synchronous triggers based migration

Original table
- insert
- delete
- update
- replace

Ghost table
- delete
- replace

pt-online-schema-change
oak-online-alter-table
LHM-ost
Triggerless, binlog based migration

- insert
- delete
- update

original table

no triggers

ghost table

binary log

gh-ost

How people build software
Binlog based design implications

- Binary logs can be read from anywhere
  - `gh-ost` prefers connecting to a replica, offloading work from master
- `gh-ost` controls the entire data flow
  - It can truly throttle, suspending all writes on the migrated server
- `gh-ost` writes are decoupled from the master workload
  - Write concurrency on master turns irrelevant
- `gh-ost`’s design is to issue all writes sequentially
  - Completely avoiding locking contention
  - Migrated server only sees a single connection issuing writes
  - Migration algorithm simplified
Binlog based migration, utilize replica
gh-ost testing

- gh-ost works perfectly well on our data
- Tested, re-tested, and tested again
- Full coverage of production tables
gh-ost testing servers

• Dedicated servers that run continuous tests
gh-ost testing replicas

production replicas

master

testing replica

production replicas

master

testing replica
gh-ost testing

- Trivial ENGINE=INNODB migration
- Stop replication
- Cut-over, cut-back
- Checksum both tables, compare
- Checksum failure: stop the world, alert
- Success/failure: event
- Drop ghost table
- Catch up
- Next table
gh-ost development cycle

• Work on branch
  .deploy gh-ost/mybranch to prod/mysql_role=ghost_testing
• Let continuous tests run
• Depending on nature of change, observe hours/days/more.
• Merge
• Tests run regardless of deployed branch
How people build so...
MySQL setup @ GitHub

- Plain-old single writer master-replicas
- Semi-sync
- Cross DC, multiple data centers
- 5.7, RBR
- Servers with special roles: production replica, backup, migration-test, analytics, ...
- 2-3 tiers of replication
- Occasional cluster split (functional sharding)
- Very dynamic, always changing
Points of failure

- Master failure, sev1
- Intermediate masters failure
orchestrator

- Topology discovery
- Refactoring
- Failovers for masters and intermediate masters
- Open source, Apache 2 license
- [github.com/github/orchestrator](https://github.com/github/orchestrator)
orchestrator failovers @ GitHub

- Automated master & intermediate master failovers for all clusters.
- On failover, runs GitHub-specific hooks
  - Grabbing VIP/DNS
  - Updating server role
  - Kicking services (e.g. pt-heartbeat)
  - Notifying chat
  - Running puppet
Testing cluster

• Dedicated testing cluster in production
• Does not take production traffic
  • “load-test” traffic
• Resembles a production topology:
  • OS, MySQL Versions
  • Data centers
  • Server roles
  • DNS
  • Proxy
• Used for many of our deployment tests
Failover testing

- Multiple times per day:
  - Setup the cluster in desired topology layout
  - Inject failure (kill/block/reject)
  - Wait, expect recovery
  - Check topology:
    - Expect new master, correct DNS changes, replica capacity, …
  - Restore old master from backup
    - (an implicit backup/restore test)
  - “success/failure” event
Failover in production

- We expect < 30s failover
- Normal case is 10-13s
- Intermediate master failover has low impact on subset of users, depending on cluster/DC/server
- Master failover implies outage
- Planned master switchover takes a few seconds
What builds trust in failovers?

A testing environment?
Chaos testing in production

- First steps into regular testing
- Manual
- Supported by our peers
- Learning, understanding impact
Tests that go wrong

• Many things can go wrong
  • Corrupt replication
  • Invalidated servers
  • Unassigned DNS
• Cleanups
Conclusion

- Backup & restore
- Failovers
- Schema migrations
Thank you!

Questions?

Jonah Berquist @hashtagjonah
Tom Krouper @CaptainEyesight