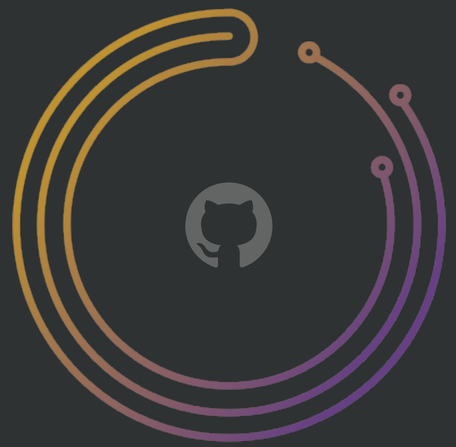
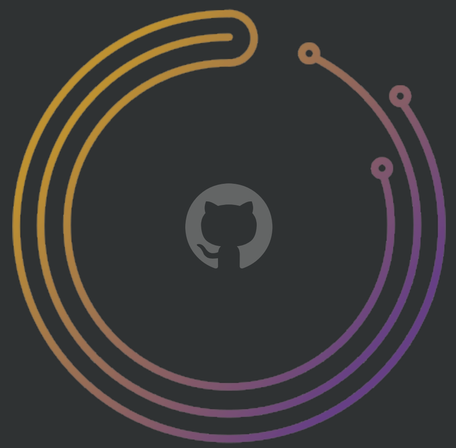


Practical Orchestrator

Shlomi Noach

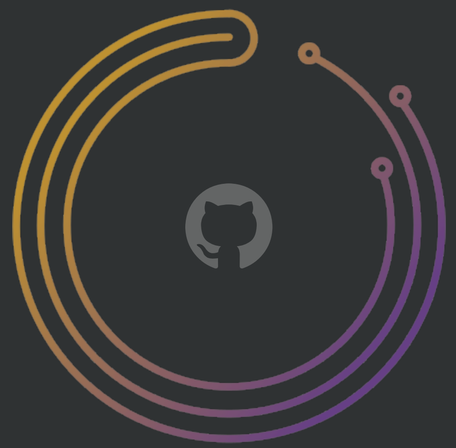
GitHub

Percona Live Europe 2017



Agenda

- Setting up orchestrator
  - Backend
  - Discovery
  - Refactoring
  - Detection & recovery
- Scripting
- HA
  - Raft cluster
- Deployment
- Roadmap



About me

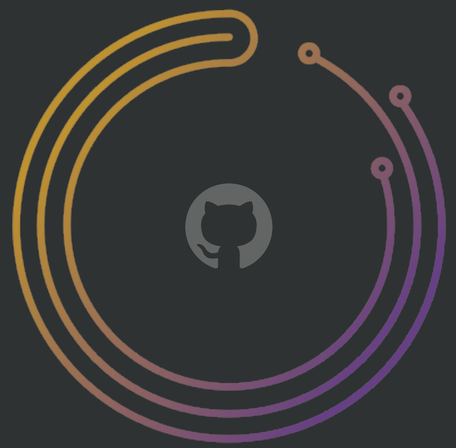
- Infrastructure engineer at GitHub
- Member of the database-infrastructure team
- MySQL community member
- Author of orchestrator, gh-ost, common_schema, freno, ccql and other open source tools.
- Blog at openark.org

github.com/shlomi-noach
@ShlomiNoach

GitHub

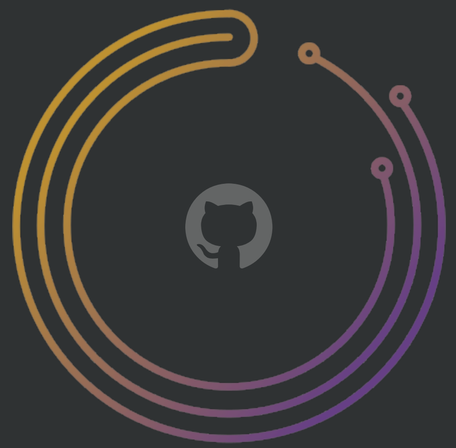
- The world's largest Octocat T-shirt and stickers store
- And water bottles
- And hoodies
- We also do stuff related to things

MySQL at GitHub



- GitHub stores repositories in git, and uses MySQL as the backend database for all related metadata:
  - Repository metadata, users, issues, pull requests, comments etc.
  - Website/API/Auth/more all use MySQL.
- We run a few (growing number of) clusters, totaling around 100 MySQL servers.
- The setup isn't very large but very busy.
- Our MySQL service must be highly available.

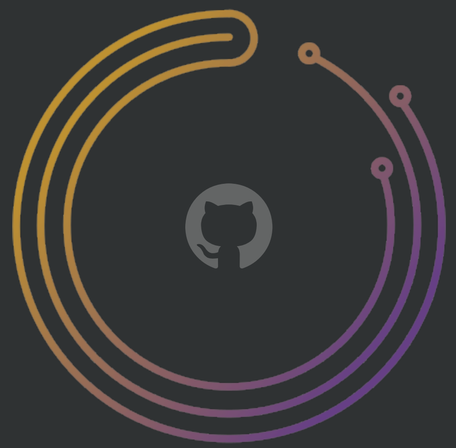
Orchestrator, meta



- Born, open sourced at Outbrain
- Further development at Booking.com, main focus on failure detection & recovery
- Adopted, maintained & supported by GitHub, github.com/github/orchestrator

- Orchestrator is free and open source, released under the Apache 2.0 license
  github.com/github/orchestrator/releases

Orchestrator



- **Discovery**

  Probe, read instances, build topology graph, attributes, queries

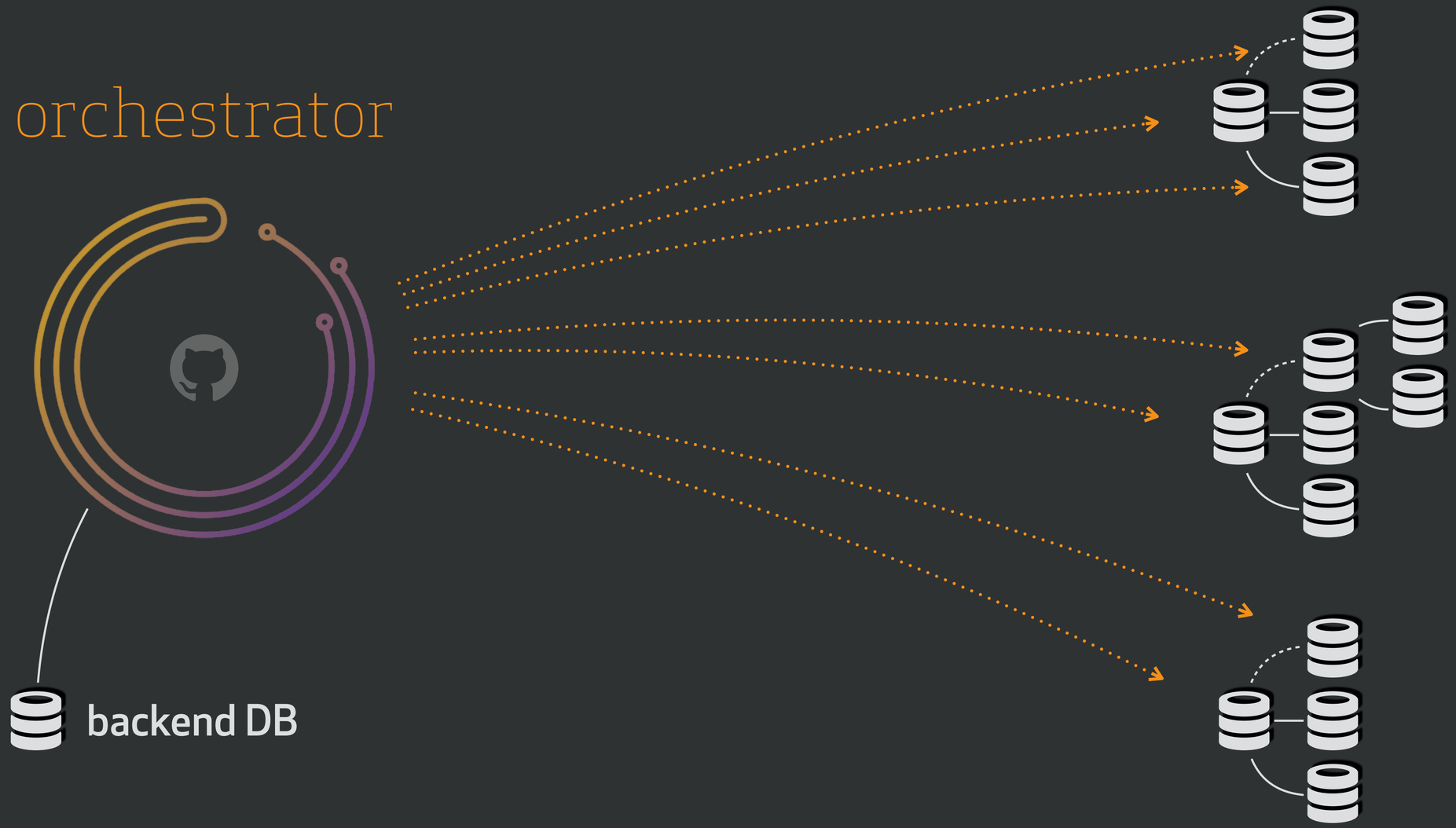
- **Refactoring**

  Relocate replicas, manipulate, detach, reorganize

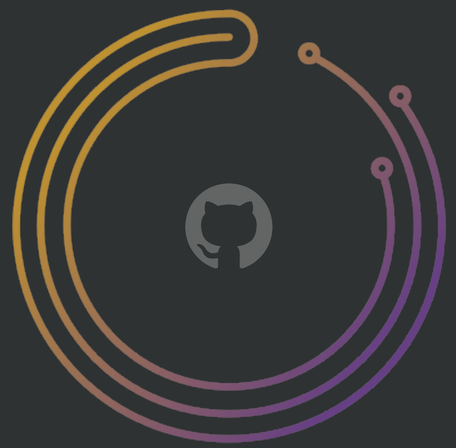
- **Recovery**

  Analyze, detect crash scenarios, structure warnings, failovers, promotions, acknowledgements, flap control, downtime, hooks

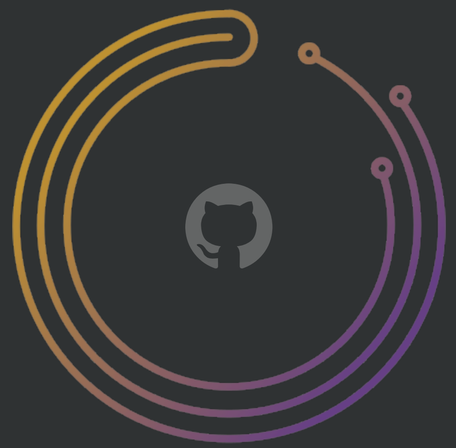
Deployment in a nutshell



Deployment in a nutshell



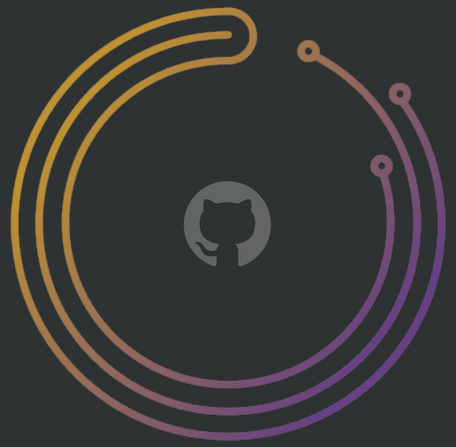
- orchestrator runs as a service
- It is *mostly stateless* (except for pending operations)
- State is stored in backend DB (MySQL/SQLite)
- orchestrator continuously discovers/probes MySQL topology servers
- Connects as client over MySQL protocol
  - Agent-less (though an agent design exists)



Agenda

- Setting up orchestrator
  - Backend
  - Discovery
  - Refactoring
  - Detection & recovery
- Scripting
- HA
  - Raft cluster
- Deployment
- Roadmap

Basic & backend setup

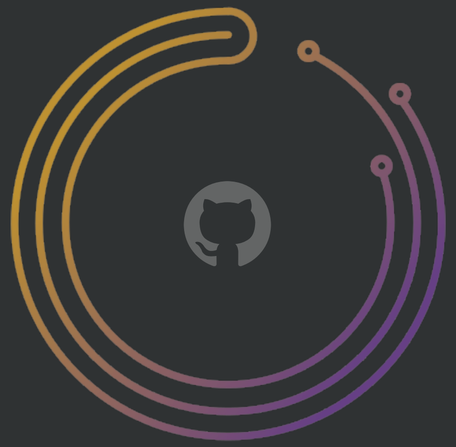


- Let orchestrator know where to find backend database
- Backend can be MySQL or SQLite

- MySQL configuration sample
- Serve HTTP on :3000

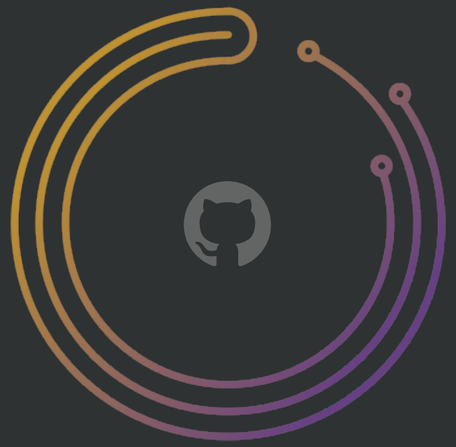
```
{
  "Debug": false,
  "ListenAddress": ":3000",

  "MySQLOrchestratorHost": "orchestrator.backend.master.com",
  "MySQLOrchestratorPort": 3306,
  "MySQLOrchestratorDatabase": "orchestrator",
  "MySQLOrchestratorCredentialsConfigFile": "/etc/mysql/orchestrator-backend.cnf",
}
```

Grants on MySQL backend



```
CREATE USER 'orchestrator_srv'@'orc_host' IDENTIFIED BY 'orc_server_password';
GRANT ALL ON orchestrator.* TO 'orchestrator_srv'@'orc_host';
```

SQLite backend



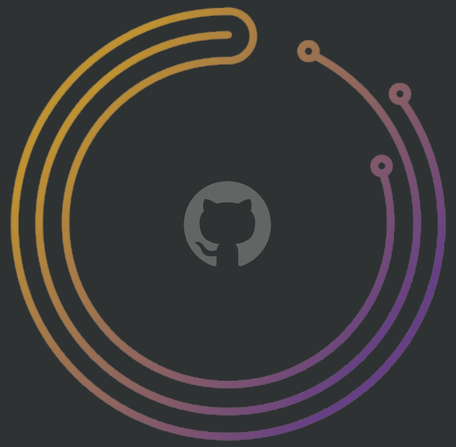
- Only applicable for:
  - standalone setups (dev, testing)
  - Raft setup (discussed later)
- Embedded with orchestrator.
- No need for MySQL backend. No backend credentials.

```
{
  "BackendDB": "sqlite",
  "SQLite3DataFile": “/var/lib/orchestrator/orchestrator.db”,
}
```

Agenda

- Setting up orchestrator
  - Backend
  - Discovery
  - Refactoring
  - Detection & recovery
- Scripting
- HA
  - Raft cluster
- Deployment
- Roadmap

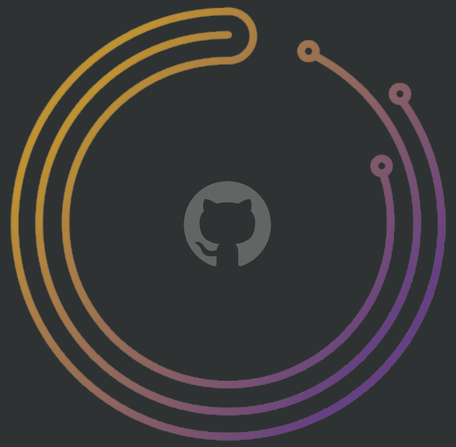
Discovery: polling servers



- Provide credentials
- Orchestrator will crawl its way and figure out the topology
- SHOW SLAVE HOSTS **requires** report_host **and** report_port **on servers**

```
{
  "MySQLTopologyCredentialsConfigFile": "/etc/mysql/orchestrator-topology.cnf",
  "InstancePollSeconds": 5,
  "DiscoverByShowSlaveHosts": false,
}
```

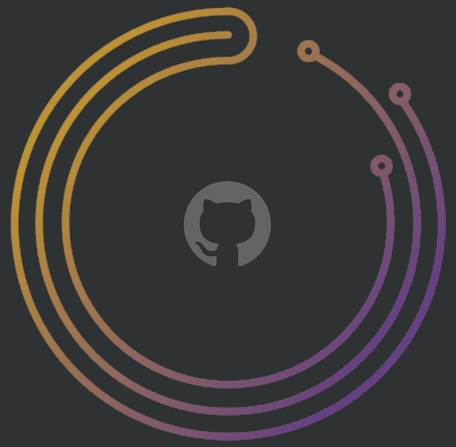
Discovery: polling servers



- Or, plaintext credentials

```
{
  "MySQLTopologyUser": "wallace",
  "MySQLTopologyPassword": "grom1t",
}
```

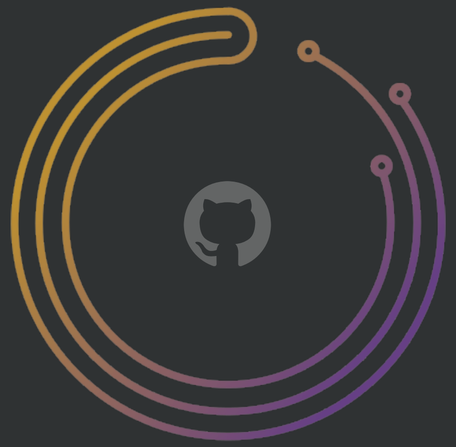
Grants on topologies



- meta **schema to be used shortly**

```
CREATE USER 'orchestrator'@'orc_host' IDENTIFIED BY 'orc_topology_password';
GRANT SUPER, PROCESS, REPLICATION SLAVE, REPLICATION CLIENT, RELOAD ON *.* TO
'orchestrator'@'orc_host';
GRANT SELECT ON meta.* TO 'orchestrator'@'orc_host';
```

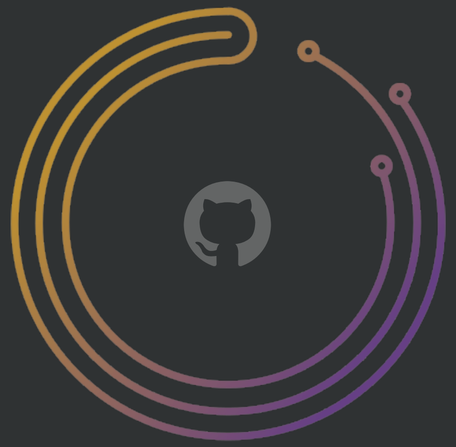
Discovery: name resolve



- Resolve & normalize hostnames
  - via DNS
  - via MySQL

```
{
  "HostnameResolveMethod": "default",
  "MySQLHostnameResolveMethod": "@@hostname"
}
```

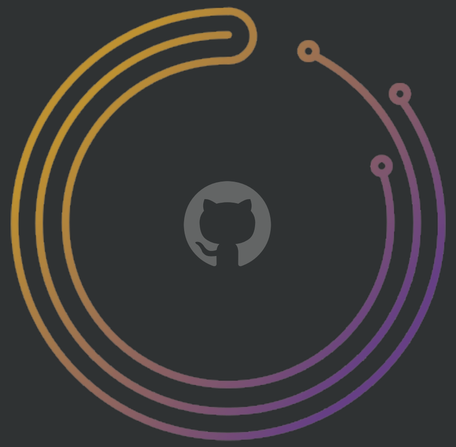
Discovery: classifying servers



- Which cluster?
- Which data center?
  - By hostname regexp or by query
- Custom replication lag query

```
{
  "ReplicationLagQuery": "select absolute_lag from meta.heartbeat_view",
  "DetectClusterAliasQuery": "select ifnull(max(cluster_name), '') as cluster_alias
from meta.cluster where anchor=1",
  "DetectClusterDomainQuery": "select ifnull(max(cluster_domain), '') as
cluster_domain from meta.cluster where anchor=1",
  "DataCenterPattern": "",
  "DetectDataCenterQuery": "select substring_index(substring_index(@@hostname, '-',
3), '-', -1) as dc",
  "PhysicalEnvironmentPattern": "",
}
```

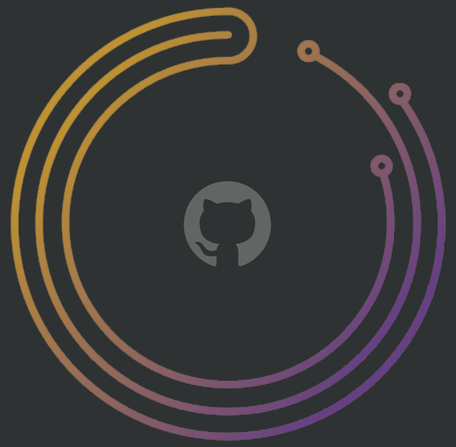
Discovery: populating cluster info



- Use meta schema
- Populate via puppet

```
CREATE TABLE IF NOT EXISTS cluster (
  anchor TINYINT NOT NULL,
  cluster_name VARCHAR(128) CHARSET ascii NOT NULL DEFAULT '',
  cluster_domain VARCHAR(128) CHARSET ascii NOT NULL DEFAULT '',
  PRIMARY KEY (anchor)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

mysql meta -e "INSERT INTO cluster (anchor, cluster_name, cluster_domain) \
  VALUES (1, '${cluster_name}', '${cluster_domain}') \
  ON DUPLICATE KEY UPDATE \
    cluster_name=VALUES(cluster_name), cluster_domain=VALUES(cluster_domain)"
```

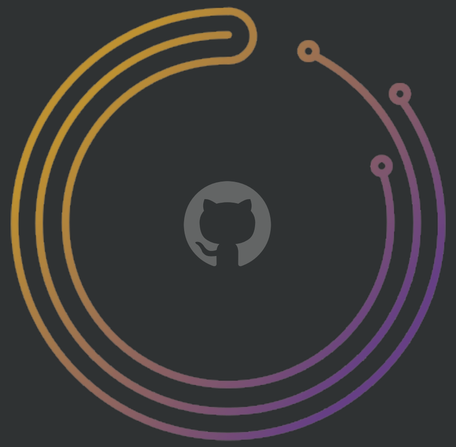
Pseudo-GTID



- Injecting Pseudo-GTID by issuing no-op DROP VIEW statements, detected both in SBR and RBR
  - This isn't visible in table data
- Possibly updating a meta table to learn about Pseudo-GTID updates.

```
set @pseudo_gtid_hint := concat_ws(':', lpad(hex(unix_timestamp(@now)), 8, '0'),
  lpad(hex(@connection_id), 16, '0'), lpad(hex(@rand), 8, '0'));
set @_pgtid_statement := concat('drop ', 'view if exists `meta`.`_pseudo_gtid_',
  'hint__asc:', @pseudo_gtid_hint, '`');
prepare st FROM @_pgtid_statement; execute st; deallocate prepare st;

insert into meta.pseudo_gtid_status (
     anchor, ..., pseudo_gtid_hint
  ) values (1, ..., @pseudo_gtid_hint)
  on duplicate key update ...
    pseudo_gtid_hint = values(pseudo_gtid_hint)
```

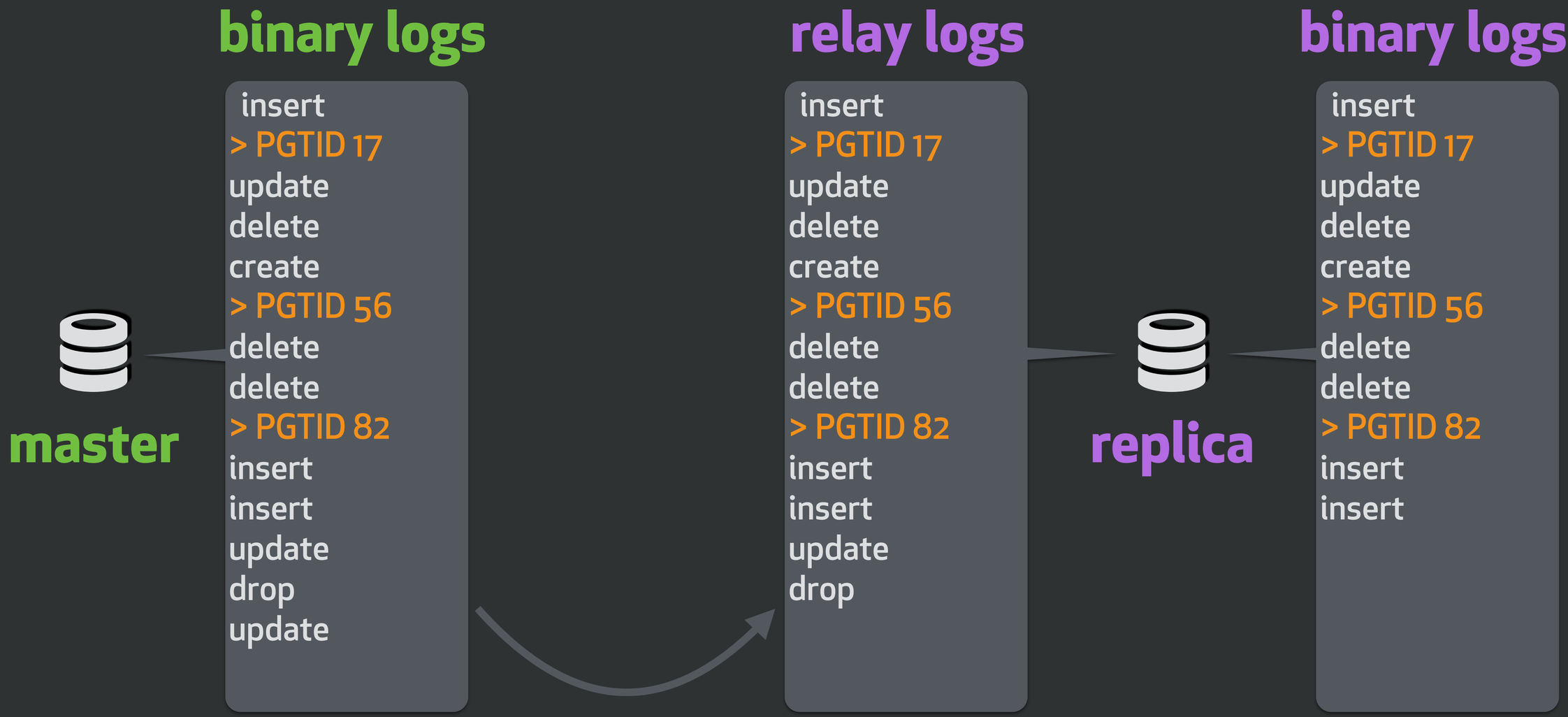
Pseudo-GTID



- Identifying Pseudo-GTID events in binary/relay logs
- Heuristics for optimized search
- Meta table lookup to heuristically identify Pseudo-GTID is available

```
{
  "PseudoGTIDPattern": "drop view if exists `meta`.`_pseudo_gtid_hint__asc:",
  "PseudoGTIDPatternIsFixedSubstring": true,
  "PseudoGTIDMonotonicHint": "asc:",
  "DetectPseudoGTIDQuery": "select count(*) as pseudo_gtid_exists
    from meta.pseudo_gtid_status
    where anchor = 1 and time_generated > now() - interval 2 hour",
}
```

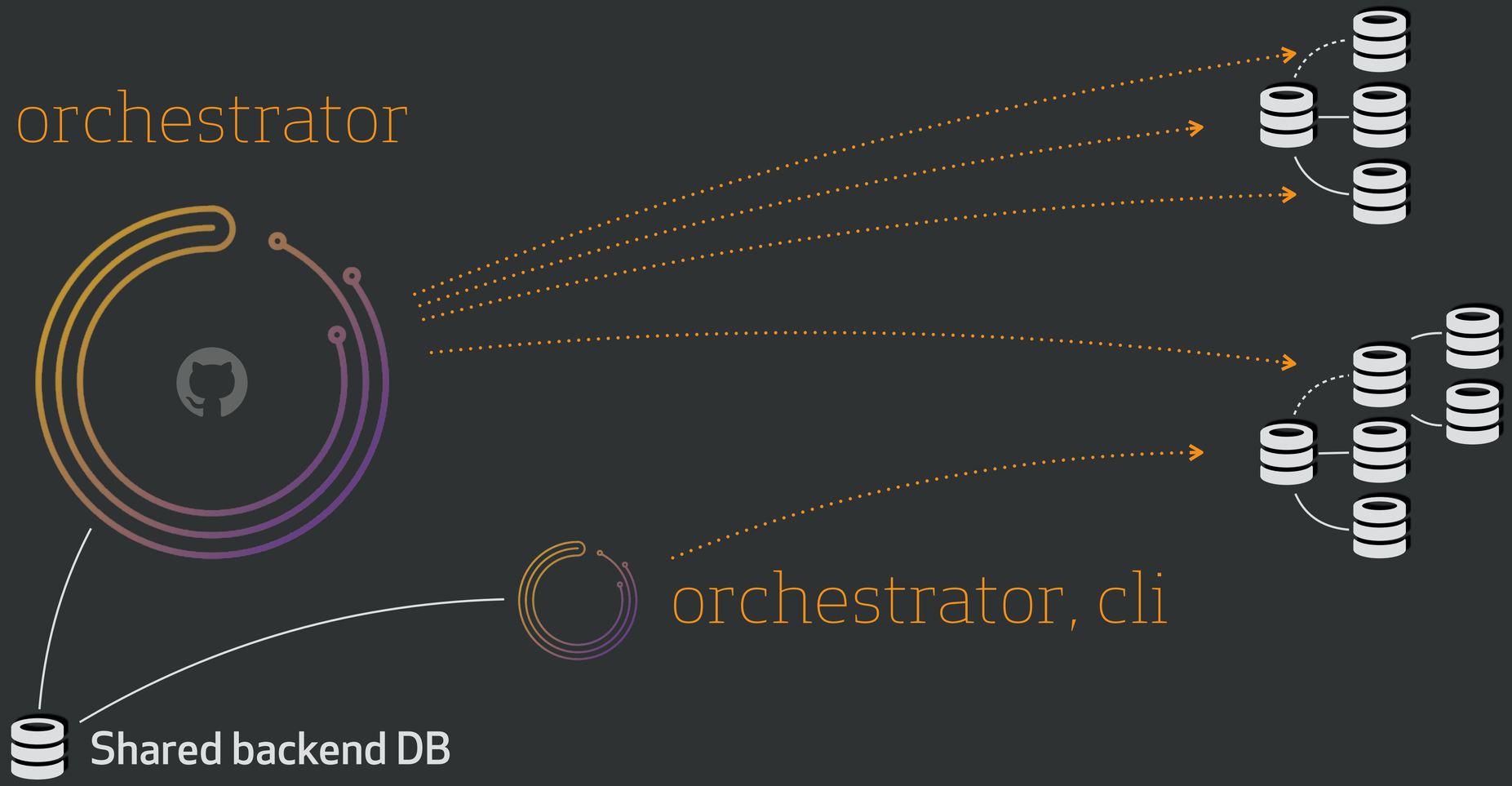
Pseudo GTID


binary logs
relay logs
binary logs
insert
> PGTID 17
update
delete
create
> PGTID 56
delete
delete
> PGTID 82
insert
insert
update
drop
update
master
replica

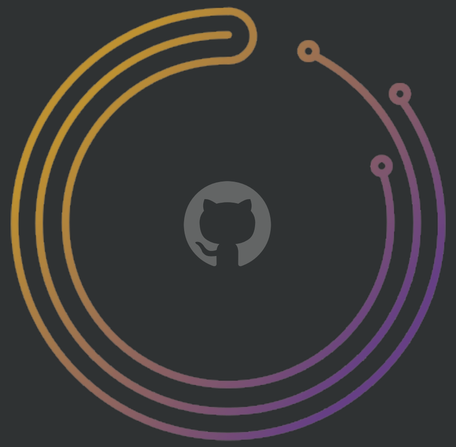
Running from command line

- Scripts, cron jobs, automation and manual labor all benefit from executing orchestrator from the command line.
- Depending on our deployment, we may choose **orchestrator-client** or the **orchestrator** binary
  - Discussed in depth later on
  - Spoiler: **orchestrator** CLI binary only supported on shared backend. orchestrator/raft requires **orchestrator-client**.
- The two have similar interface.

Deployment, CLI



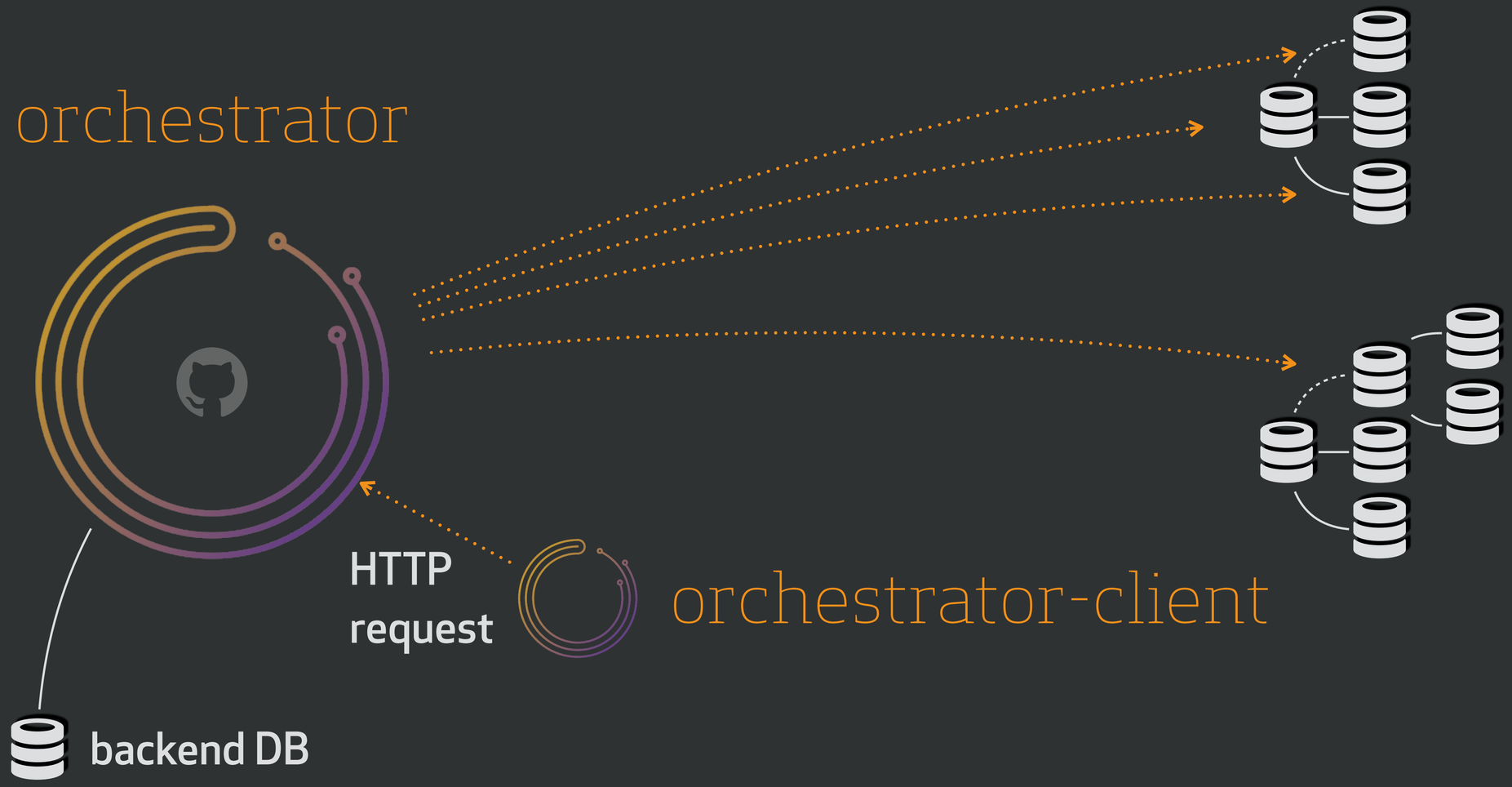
CLI



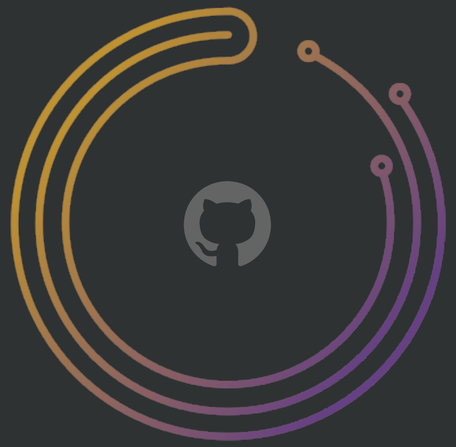
- Connects to same backend DB as the orchestrator service

```
orchestrator
orchestrator -c help

Available commands (-c):
Smart relocation:
       relocate              Relocate a replica beneath another instance
       relocate-replicas     Relocates all or part of the replicas of a given
Information:
       clusters              List all clusters known to orchestrator
```

Deployment, orchestrator-client



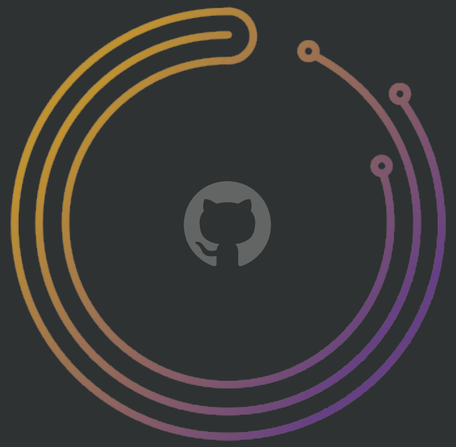
orchestrator-client



- Connects to orchestrator service node via API
- Analyzes JSON response, parses as needed
- Provides command-line interface similar to orchestrator CLI

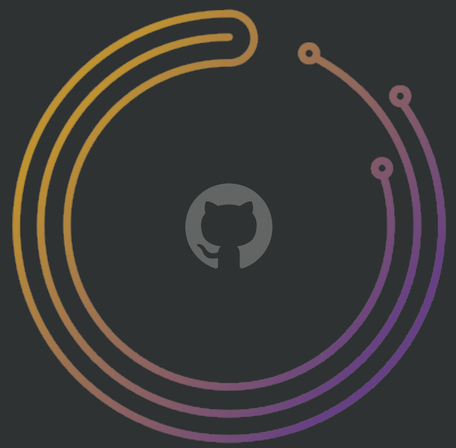
```
orchestrator-client
orchestrator-client -c help

Usage: orchestrator-client -c <command> [flags...]
Example: orchestrator-client -c which-master -i some.replica
Available commands:
   discover    Lookup an instance, investigate it
   forget      Forget about an instance's existence
   clusters    List all clusters known to orchestrator
   relocate    Relocate a replica beneath another instance
   recover     Do auto-recovery given a dead instance, …
```

client: information



- What kind of information can we pull having discovered our topologies?

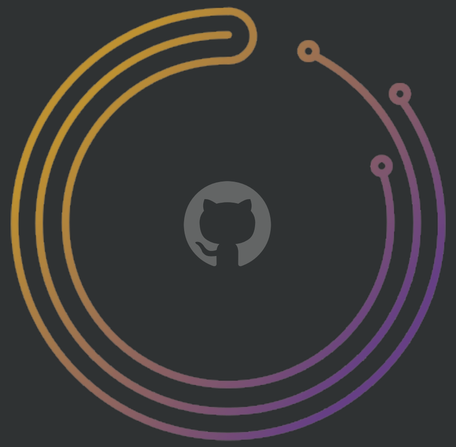
```
orchestrator-client -c clusters
orchestrator-client -c all-instances

orchestrator-client -c which-cluster some.instance.in.cluster
orchestrator-client -c which-cluster-instances -alias mycluster
orchestrator-client -c which-master some.instance
orchestrator-client -c which-replicas some.instance

orchestrator-client -c topology -alias mycluster
```



Agenda

- Setting up orchestrator
  - Backend
  - Discovery
  - Refactoring
  - Detection & recovery
- Scripting
- HA
  - Raft cluster
- Deployment
- Roadmap

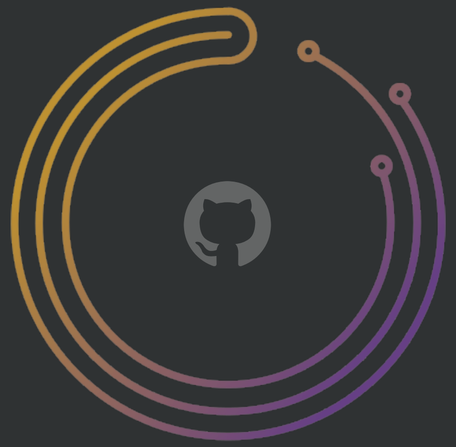
client: refactoring



- Smart: let orchestrator figure out *how* to refactor:
  - GTID
  - Pseudo-GTID
  - Normal file:pos

```
orchestrator-client -c relocate
  -i which.instance.to.relocate -d instance.below.which.to.relocate

orchestrator-client -c relocate-replicas
  -i instance.whose.replicas.to.relocate -d instance.below.which.to.relocate
```

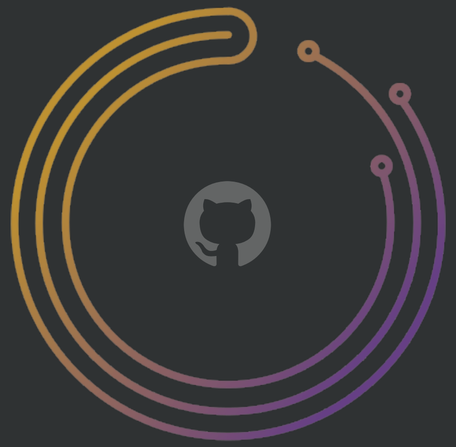
client: refactoring



- file:pos specific

```
orchestrator-client -c move-below
  -i which.instance.to.relocate -d instance.below.which.to.relocate

orchestrator-client -c move-up -i instance.to.move
```

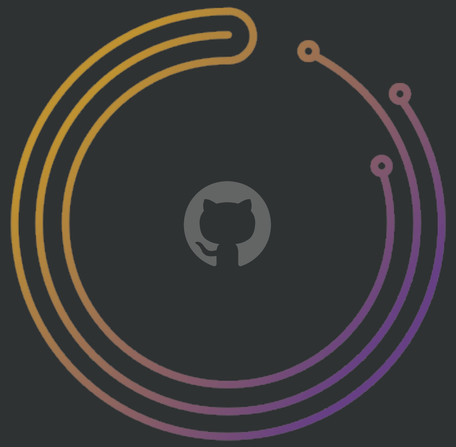
orchestrator-client: various commands



- Using -c detach-replica to intentionally break replication, in a reversible way

```
orchestrator-client -c set-read-only -i some.instance.com
orchestrator-client -c set-writeable -i some.instance.com

orchestrator-client -c stop-slave -i some.instance.com
orchestrator-client -c start-slave -i some.instance.com
orchestrator-client -c restart-slave -i some.instance.com
orchestrator-client -c skip-query -i some.instance.com

orchestrator-client -c detach-replica -i some.instance.com
orchestrator-client -c reattach-replica -i some.instance.com
```

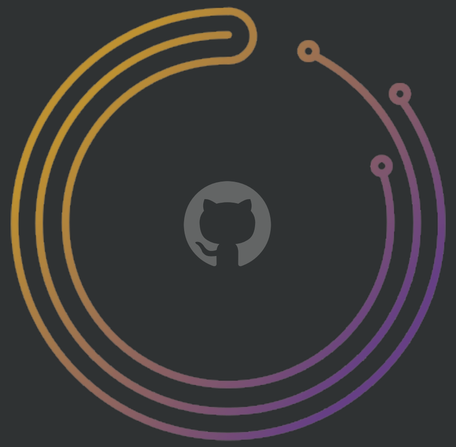
client: some fun



- Flatten a topology
- Operate on all replicas
- See also https://github.com/github/ccql
- We'll revisit shortly

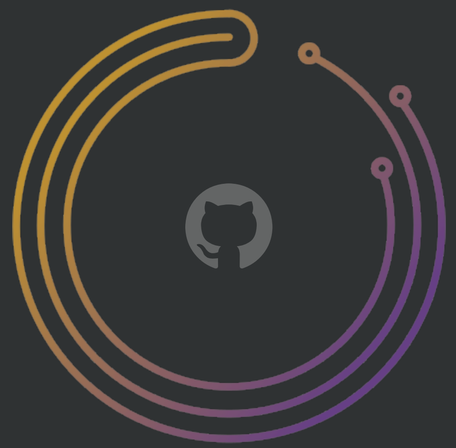
```
master=$(orchestrator-client -c which-cluster-master -alias mycluster)

orchestrator-client -c which-cluster-instances -alias mycluster | while read i ; do \
  orchestrator-client -c relocate -i $i -d $master \
done

orchestrator-client -c which-replicas -i $master | while read i ; do \
  orchestrator-client -c set-read-only -i $i \
done
```

API



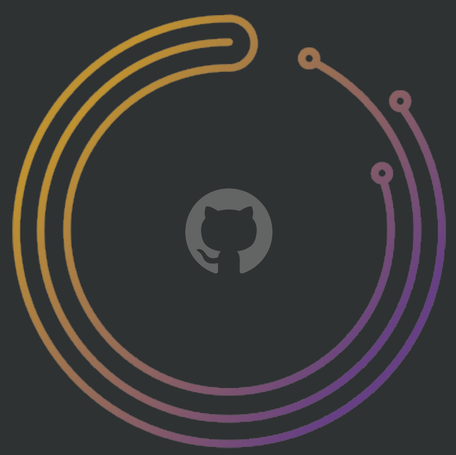
- The web interface is merely a facade for API calls
- orchestrator-client uses the API behind the scenes
- The API is powerful and full of information

```
curl -s "http://localhost:3000/api/cluster/alias/mycluster" | jq .

curl -s “http://localhost:3000/api/instance/some.host/3306" | jq .

curl -s “http://localhost:3000/api/relocate/some.host/3306/another.host/3306” | jq .
```



Agenda

- Setting up orchestrator
  - Backend
  - Discovery
  - Refactoring
  - Detection & recovery
- Scripting
- HA
  - Raft cluster
- Deployment
- Roadmap



Detection & recovery primer

- What's so complicated about detection & recovery?
- How is orchestrator different than other solutions?
- What makes a reliable detection?
- What makes a successful recovery?
- Which parts of the recovery does orchestrator own?
- What about the parts it doesn't own?

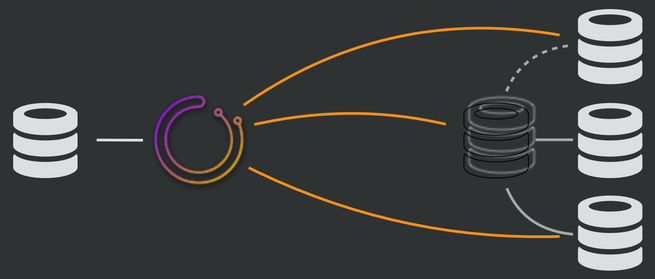
Detection

- orchestrator continuously probes all MySQL topology servers
- At time of crash, orchestrator knows what the topology *should look like*, because it knows how it looked like a moment ago
- What insights can orchestrator draw from this fact?

Other tools: dead master detection

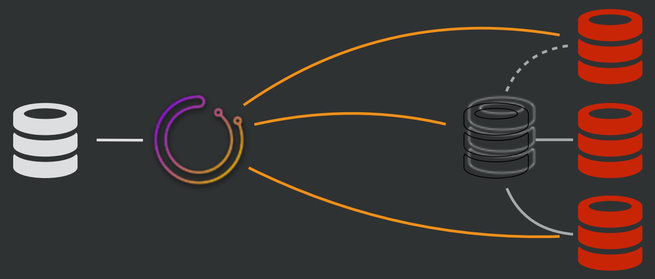
- Common failover tools only observe per-server health.
- If the master cannot be reached, it is considered to be dead.
- To avoid false positives, some introduce repetitive checks + intervals.
  - e.g. check every 5 seconds and if seen dead for 4 consecutive times, declare “death”
- This heuristically *reduces* false positives, and introduces recovery latency.

Detection: dead master, holistic approach



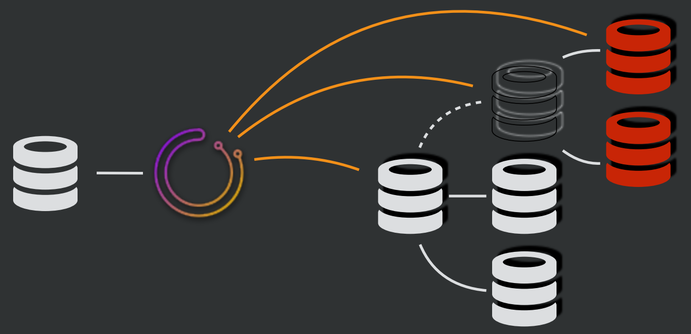
- orchestrator uses a holistic approach. It harnesses the topology itself.
- orchestrator observes the master **and** the replicas.
- If the master is unreachable, but all replicas are happy, then there's no failure. It may be a network glitch.

Detection: dead master, holistic approach



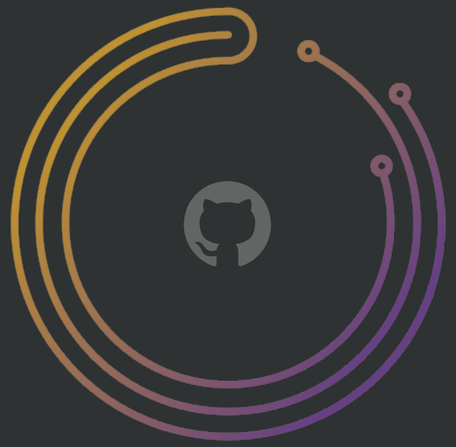
- If the master is unreachable, **and** all of the replicas are in agreement (replication broken), then declare “death”.
- There is no need for repetitive checks. Replication broke on all replicas due to a reason, and following its own timeout.

Detection: dead intermediate master



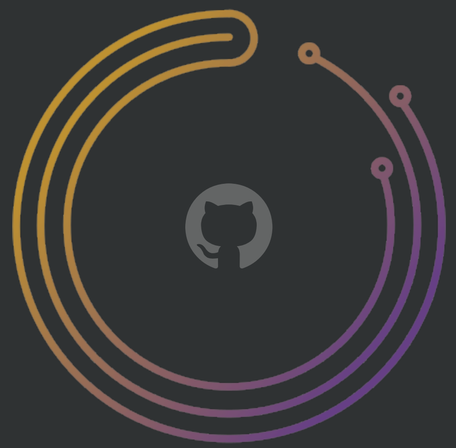
- orchestrator uses exact same holistic approach logic
- If intermediate master is unreachable **and** its replicas are broken, then declare “death”

Recovery: basic config



- How frequently to analyze/recover topologies
- Block detection interval

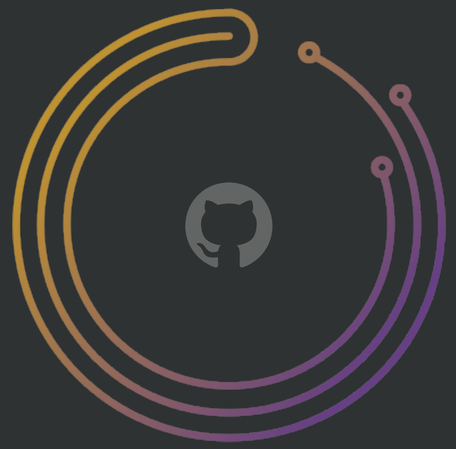
```
{
  "RecoveryPollSeconds": 2,
  "FailureDetectionPeriodBlockMinutes": 60,
}
```

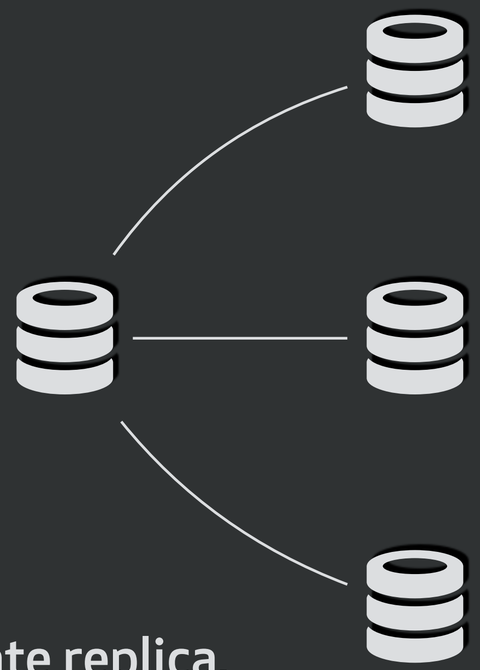


Recovery & promotion constraints

- You've made the decision to promote a new master
- Which one?
- Are all options valid?
- Is the current state what you think the current state is?

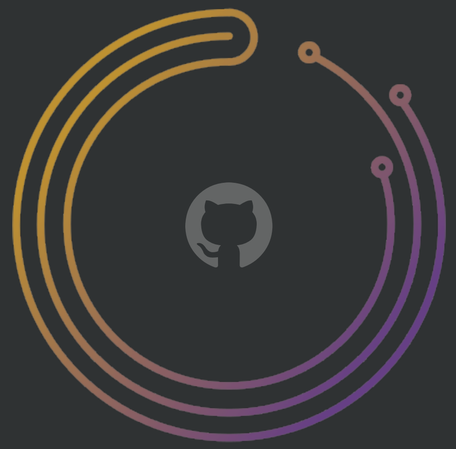
Promotion constraints

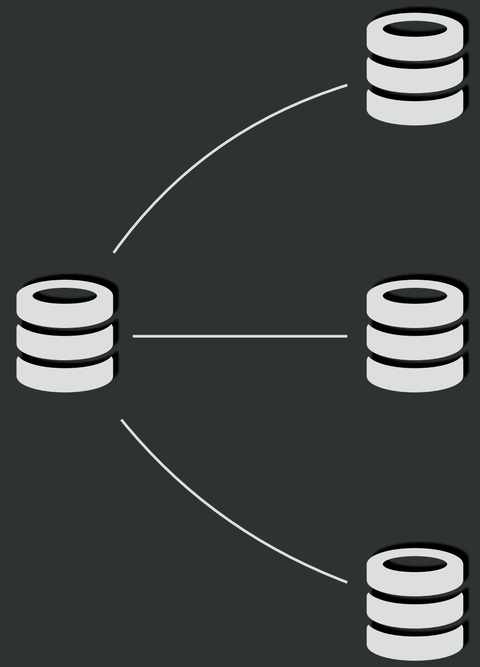




You wish to promote the most up to date replica, otherwise you give up on any replica that is more advanced

Promotion constraints





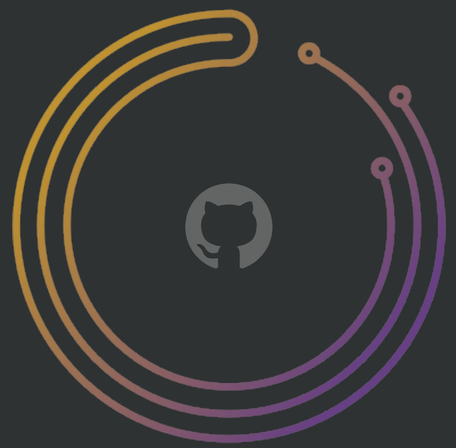
log_slave_updates

log_slave_updates

No binary logs

You must not promote a replica that has no binary logs, or without **log_slave_updates**

Promotion constraints



You prefer to promote a replica from same DC as failed master

Promotion constraints

You must not promote Row Based Replication server on top of Statement Based Replication

Promotion constraints

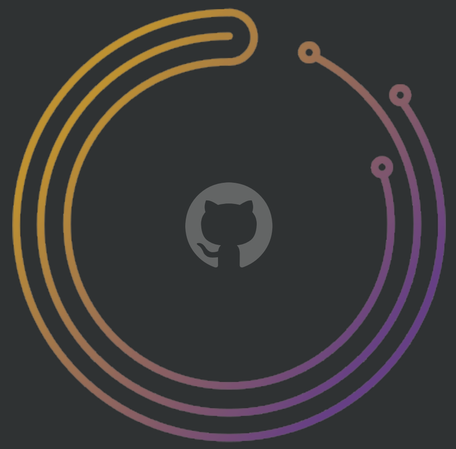
Promoting 5.7 means losing 5.6 (replication not forward compatible)
So Perhaps worth losing the 5.7 server?

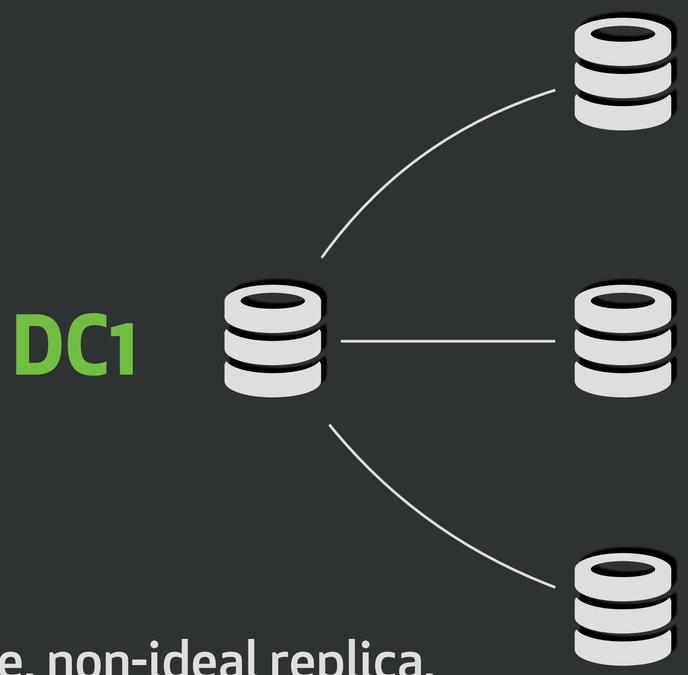
Promotion constraints

But if *most* of your servers are 5.7, and 5.7 turns to be most up to date, better promote 5.7 and drop the 5.6

Orchestrator handles this logic and prioritizes promotion candidates by overall count and state of replicas

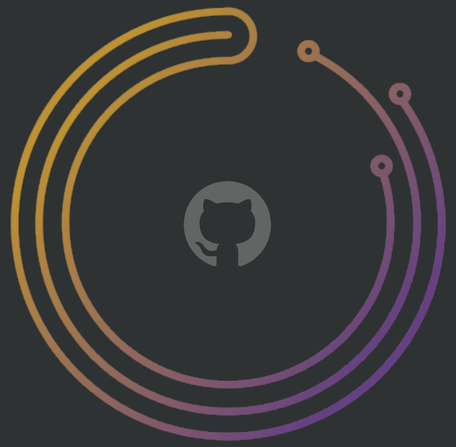
Promotion constraints, real life





Orchestrator can promote one, non-ideal replica, have the rest of the replicas converge, and then refactor again, promoting an ideal server

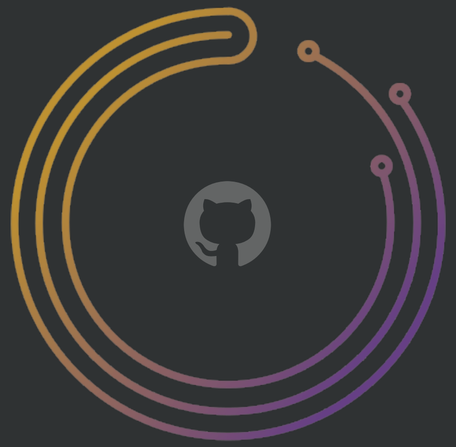
Recovery: general recovery rules



- Anti-flapping control
- Old style, hostname/regexp based promotion black list
- Which cluster to auto-failover?
  - Master / intermediate-master?

```
{
  "RecoveryPeriodBlockSeconds": 3600,
  "RecoveryIgnoreHostnameFilters": [],
  "RecoverMasterClusterFilters": [
    "thiscluster",
    "thatcluster"
  ],
  "RecoverIntermediateMasterClusterFilters": [
    "*"
  ],
}
```

client: recovery



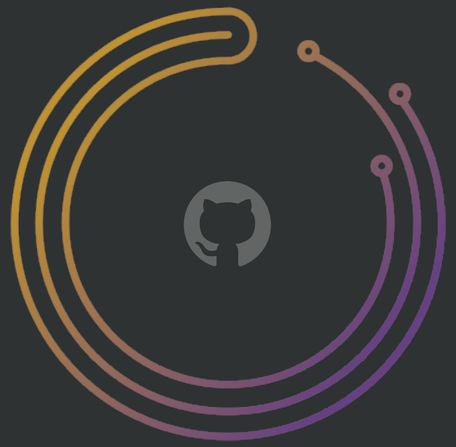
- A human may always kick in recovery even if automated recoveries are disabled for a cluster.
- A human overrides flapping considerations.

```
orchestrator-client -c replication-analysis

orchestrator-client -c recover -i a.dead.instance.com
orchestrator-client -c ack-cluster-recoveries -i a.dead.instance.com

orchestrator-client -c graceful-master-takeover -alias mycluster
orchestrator-client -c force-master-failover -alias mycluster # danger zone!

orchestrator-client -c register-candidate -i candidate.replica -promotion-rule prefer
```

Recovery: hooks

```
{
  "OnFailureDetectionProcesses": [
    "echo 'Detected {failureType} on {failureCluster}. Affected replicas:
    {countReplicas}' >> /tmp/recovery.log"
  ],
  "PreFailoverProcesses": [
    "echo 'Will recover from {failureType} on {failureCluster}' >> /tmp/recovery.log"
  ],
  "PostFailoverProcesses": [
    "echo '(for all types) Recovered from {failureType} on {failureCluster}. Failed:
    {failedHost}:{failedPort}; Successor: {successorHost}:{successorPort}' >> /tmp/
    recovery.log"
  ],
  "PostUnsuccessfulFailoverProcesses": [],
  "PostMasterFailoverProcesses": [
    "echo 'Recovered from {failureType} on {failureCluster}. Failed: {failedHost}:
    {failedPort}; Promoted: {successorHost}:{successorPort}' >> /tmp/recovery.log"
  ],
  "PostIntermediateMasterFailoverProcesses": [],
}
```

Recovery: promotion actions



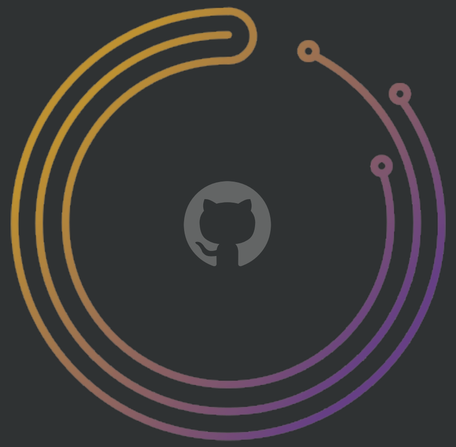
- With great power comes great configuration complexity
- Different users need different behavior

```
{
  "ApplyMySQLPromotionAfterMasterFailover": true,
  "MasterFailoverLostInstancesDowntimeMinutes": 10,
  "FailMasterPromotionIfSQLThreadNotUpToDate": true,
  "DetachLostReplicasAfterMasterFailover": true,
}
```

Agenda

- Setting up orchestrator
  - Backend
  - Discovery
  - Refactoring
  - Detection & recovery
- Scripting
- HA
  - Raft cluster
- Deployment
- Roadmap

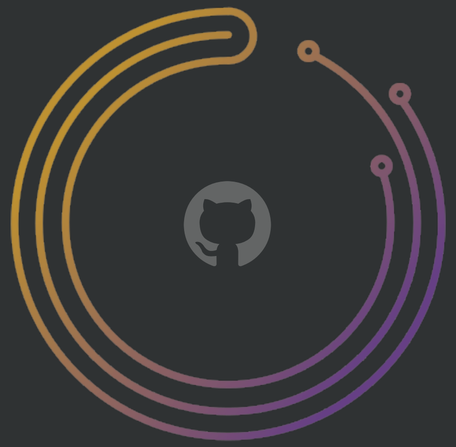
Scripting: master failover testing automation



- Preparation:
  - Flatten topology
  - Create an intermediate master with two replicas

```
master=$(orchestrator-client -c which-cluster-master -alias mycluster)

orchestrator-client -c which-cluster-instances -alias mycluster | while read i ; do \
  orchestrator-client -c relocate -i $i -d $master \
done

intermediate_master=$(orchestrator-client -c which-replicas -i $master | shuf | head -1)
orchestrator-client -c which-replicas -i $master | grep -v $intermediate_master | shuf |
head -2 | while read i ; do \
  orchestrator-client -c relocate -i $i -d $intermediate_master \
done
```

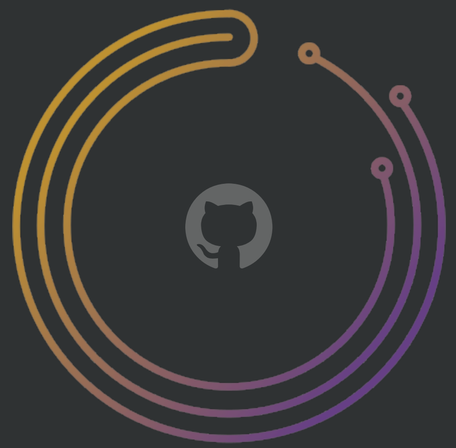
Scripting: master failover testing automation



- Kill the master, wait some time
- Expect new master
- Expect enough replicas
- Add your own tests & actions: write to master, expect data on replicas; verify replication lag; restore dead master, …

```
# kill MySQL on master...
sleep 30 # graceful wait for recovery

new_master=$(orchestrator-client -c which-cluster-master -alias mycluster)
[ -z "$new_master" ] && { echo "strange, cannot find master" ; exit 1 ; }
[ "$new_master" == "$master" ] && { echo "no change of master" ; exit 1 ; }
orchestrator-client -c which-cluster-instances -alias mycluster | while read i ; do \
  orchestrator-client -c relocate -i $i -d $new_master \
done
count_replicas=$(orchestrator-client -c which-replicas -i $new_master | wc -l)
[ $count_replicas -lt 4 ] && { echo "not enough salvaged replicas" ; exit 1 ; }
```

MySQL configuration advice

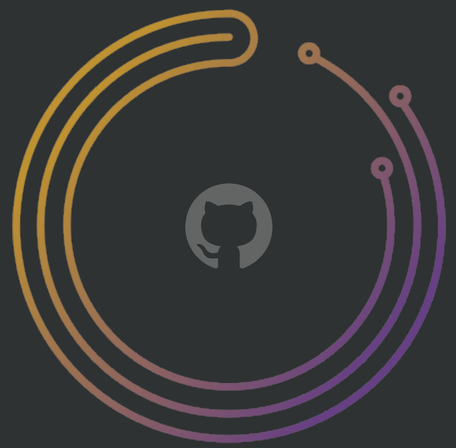


- slave_net_timeout=4
  - **Implies heartbeat period=2**
- CHANGE MASTER TO
  MASTER_CONNECT_RETRY=1,
  MASTER_RETRY_COUNT=86400
- **For Orchestrator to detect replication credentials,**
  - master_info_repository=TABLE
  - **Grants on** mysql.slave_master_info

Agenda

- Setting up orchestrator
  - Backend
  - Discovery
  - Refactoring
  - Detection & recovery
- Scripting
- HA
  - Raft cluster
- Deployment
- Roadmap

High Availability



- Orchestrator takes care of MySQL high availability. What makes orchestrator itself highly available?
- Orchestrator requires a backend database. HA for orchestrator, therefore, needs:
  - HA of the orchestrator service
  - HA of the backend DB

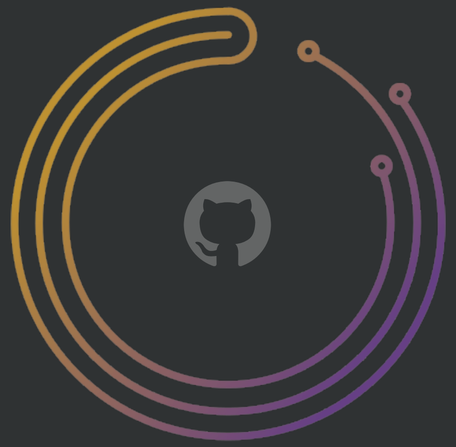
HA via shared backend (sync replication)

- Galera/XtraDB Cluster/InnoDB Cluster, multi-write mode
- 1:1 mapping between orchestrator nodes and cluster nodes
- Ideally orchestrator & MySQL run on same box
- HA achieved via synchronous replication consensus
- Orchestrator leader guaranteed to speak to MySQL quorum
- Any node can fail, service remains available

HA via raft consensus

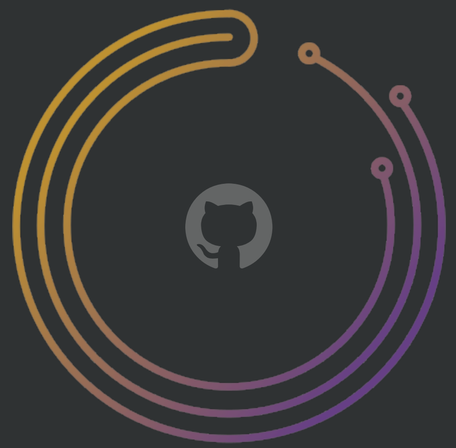
- Orchestrator runs in raft mode
- Orchestrator nodes form consensus
  - Leader guaranteed to have consensus
- Each orchestrator node has dedicated backend DB
  - MySQL, ideally on same box
  - Or SQLite, embedded
- No database replication; DBs are standalones
- Any node can fail, service remains available

orchestrator/raft setup



- Enable raft
- Specify complete list of raft nodes including this node
- 3 or 5 nodes preferable
  - Cross DC is possible and desired
- RaftBind is address of this node

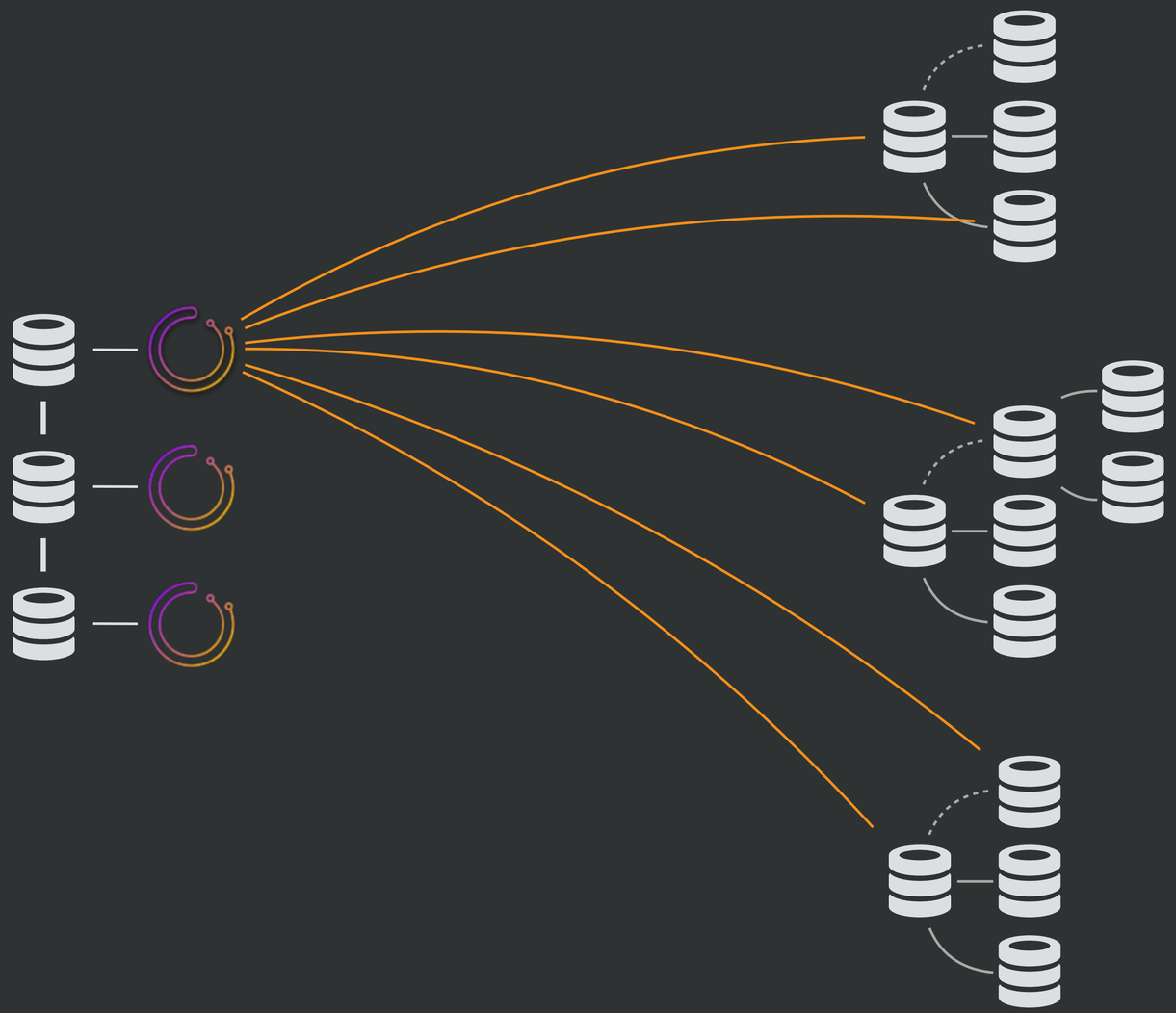
```
{
  "RaftEnabled": true,
  "RaftBind": "<ip.or.fqdn.of.this.orchestrator.node>",
  "DefaultRaftPort": 10008,
  "RaftNodes": [
    "<ip.or.fqdn.of.orchestrator.node1>",
    "<ip.or.fqdn.of.orchestrator.node2>",
    "<ip.or.fqdn.of.orchestrator.node3>"
  ],
}
```



Agenda

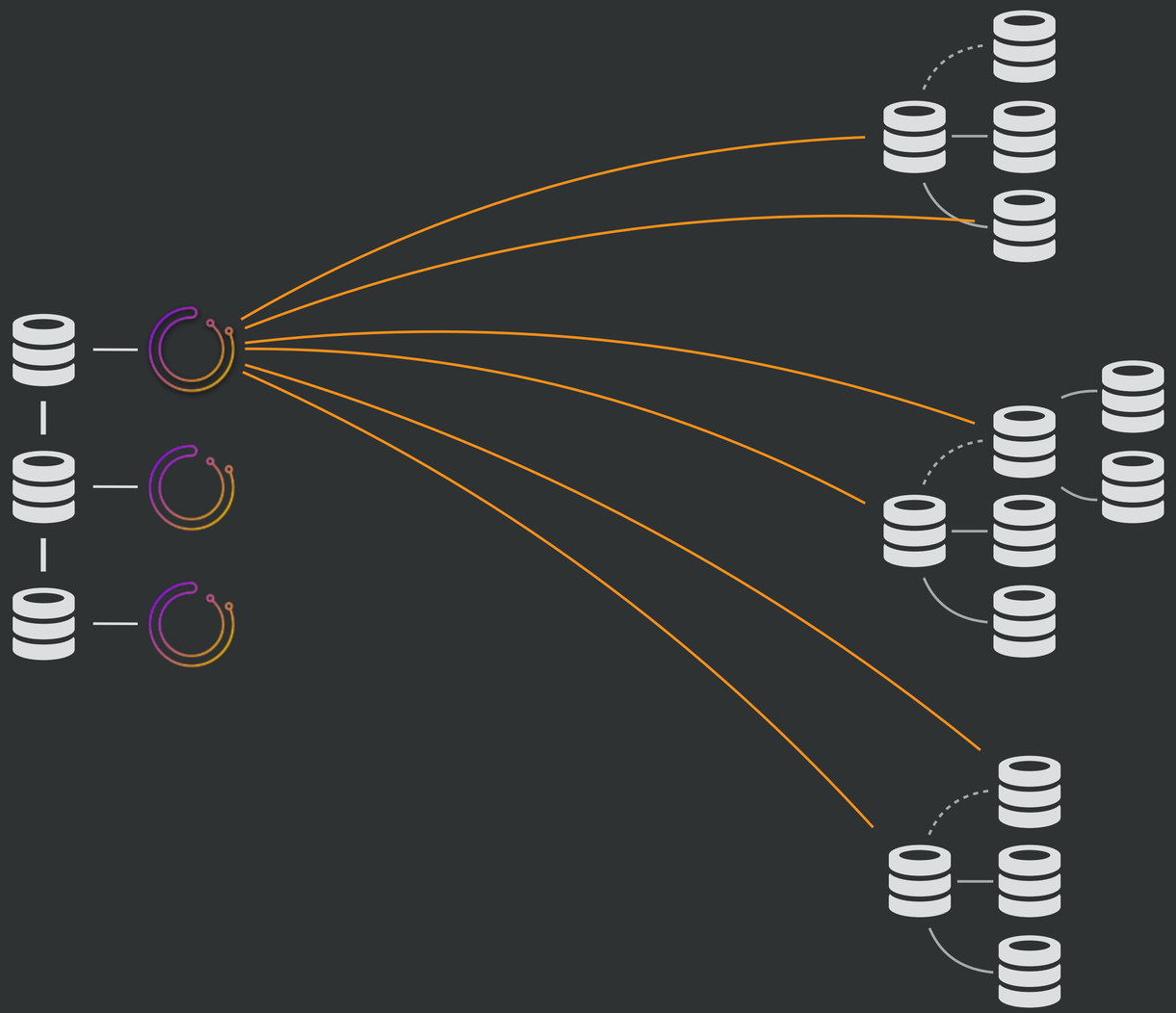
- Setting up orchestrator
  - Backend
  - Discovery
  - Refactoring
  - Detection & recovery
- Scripting
- HA
  - Raft cluster
- Deployment
- Roadmap

Shared backend deployment



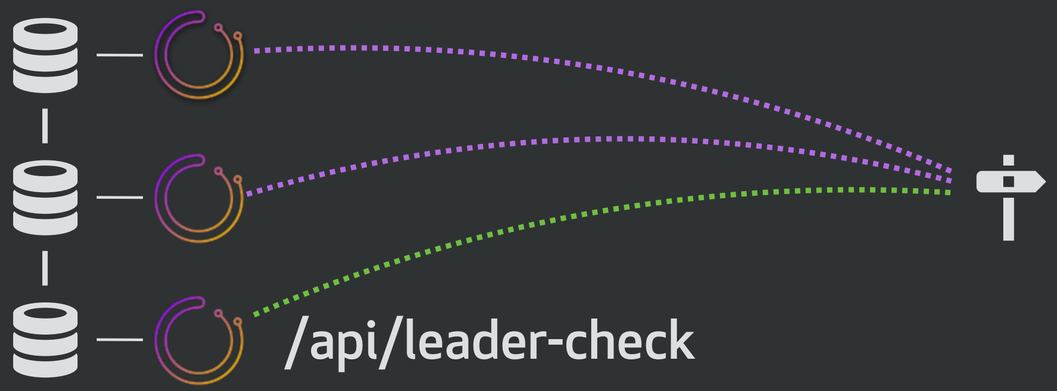
- Single orchestrator node (the leader) probes all MySQL backends
  - Roadmap: distribute probe jobs
- Data is implicitly shared to all orchestrator nodes

Shared backend deployment



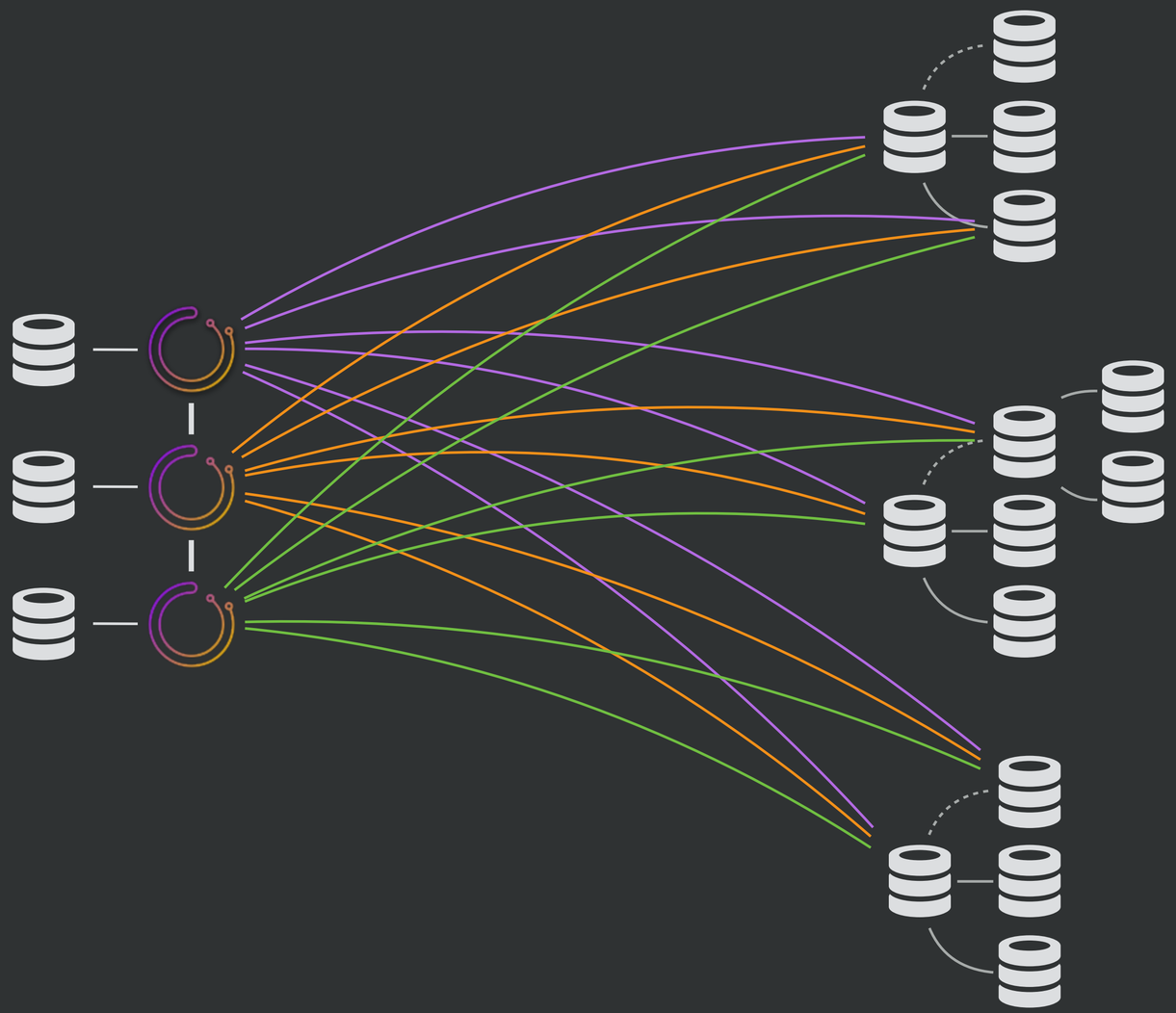
- You may speak to any **healthy** orchestrator service node
- Ideally you'd speak to the **leader** at any given time

Shared backend deployment



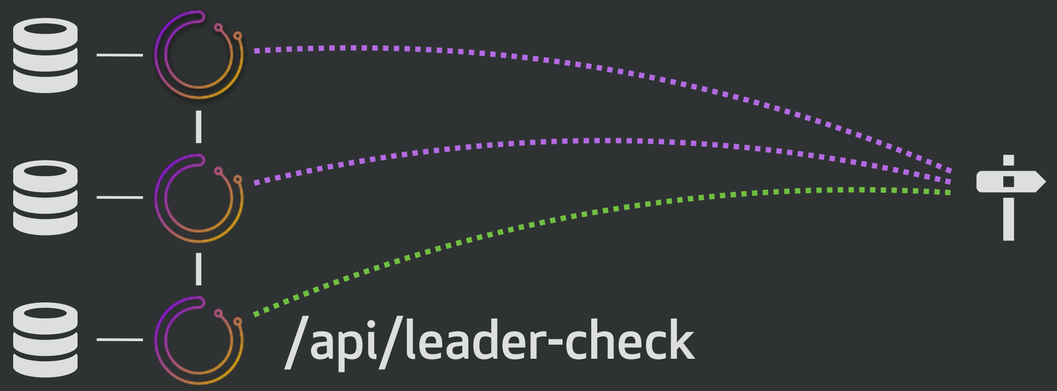
- You may choose to place a proxy in front of orchestrator nodes
- Check /api/leader-check to direct traffic to leader
- The proxy doesn't serve HA, purposes, merely convenience
- **orchestrator-client** is able to connect to leader regardless of proxy

orchestrator/raft deployment



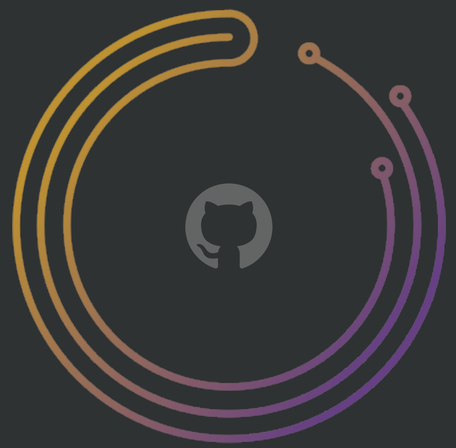
- Each orchestrator node polls all MySQL servers
  - Roadmap: distribute probe jobs
- DB backends have similar (not identical) data
- One node is leader, has quorum

orchestrator/raft deployment



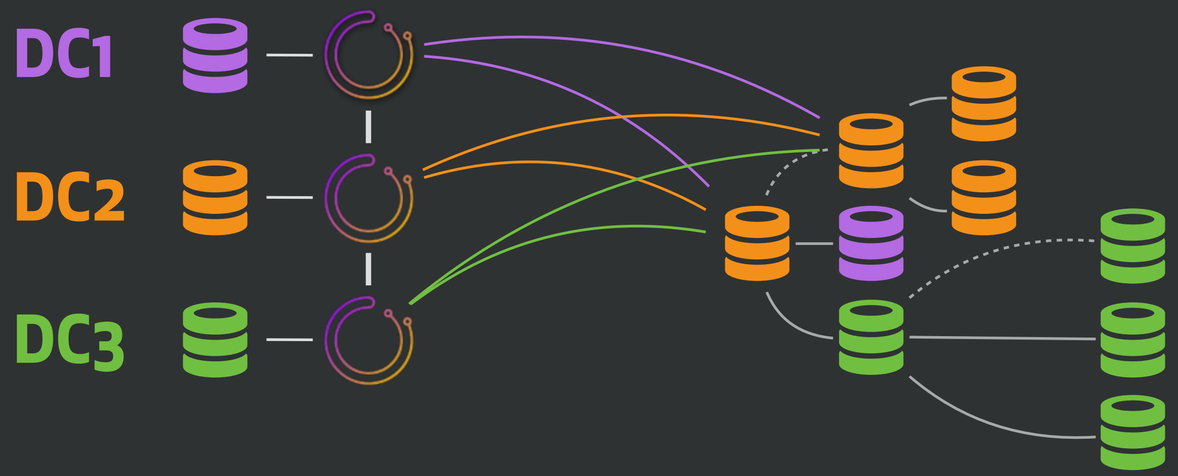
- You may only speak to the leader
- Non-leader nodes are read-only and should be avoided
- You may choose to place a proxy in front of orchestrator nodes
- Check /api/leader-check to direct traffic to leader
- The proxy doesn't serve HA, purposes, merely convenience
- **orchestrator-client** is able to connect to leader regardless of proxy

Why orchestrator/raft?



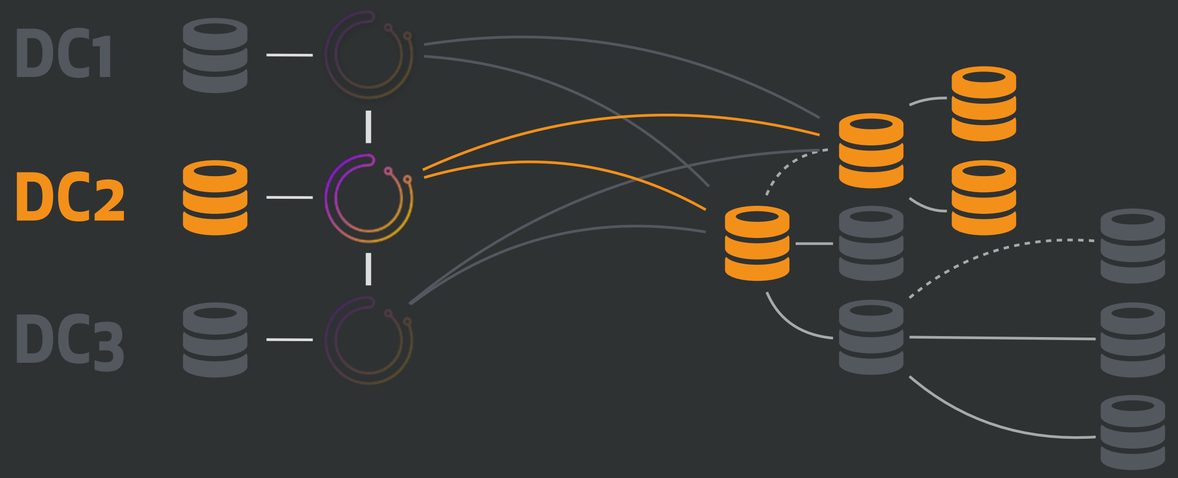
- High availability
- SQLite backend, embedded within orchestrator, allows lightweight deployments
- Handles DC fencing based on quorum

orchestrator/raft: fencing



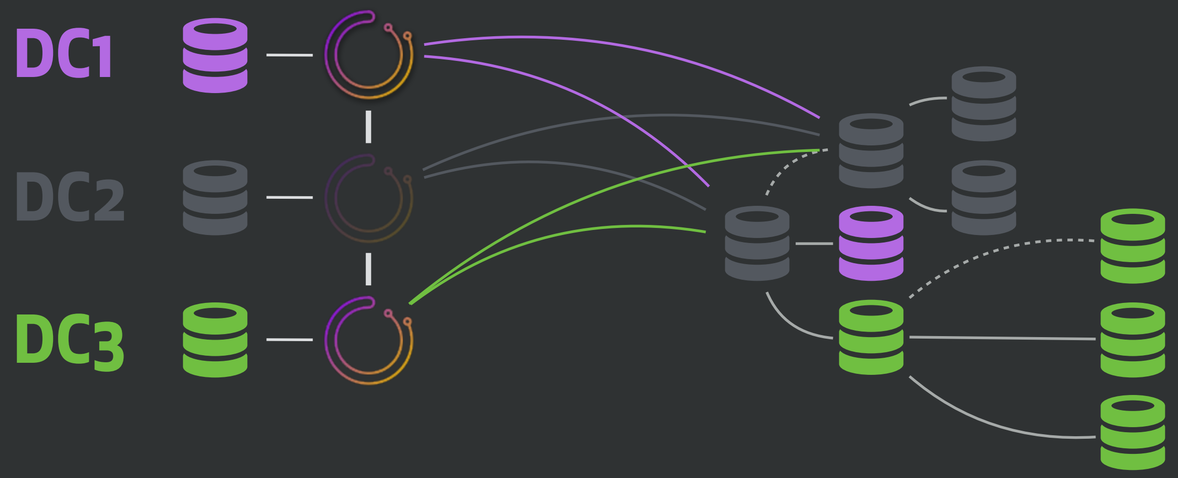
- Assume this 3 DC setup
- One orchestrator node in each DC
- Master and a few replicas in DC2
- What happens if DC2 gets network partitioned?
  - i.e. no network in or out DC2

orchestrator/raft: fencing



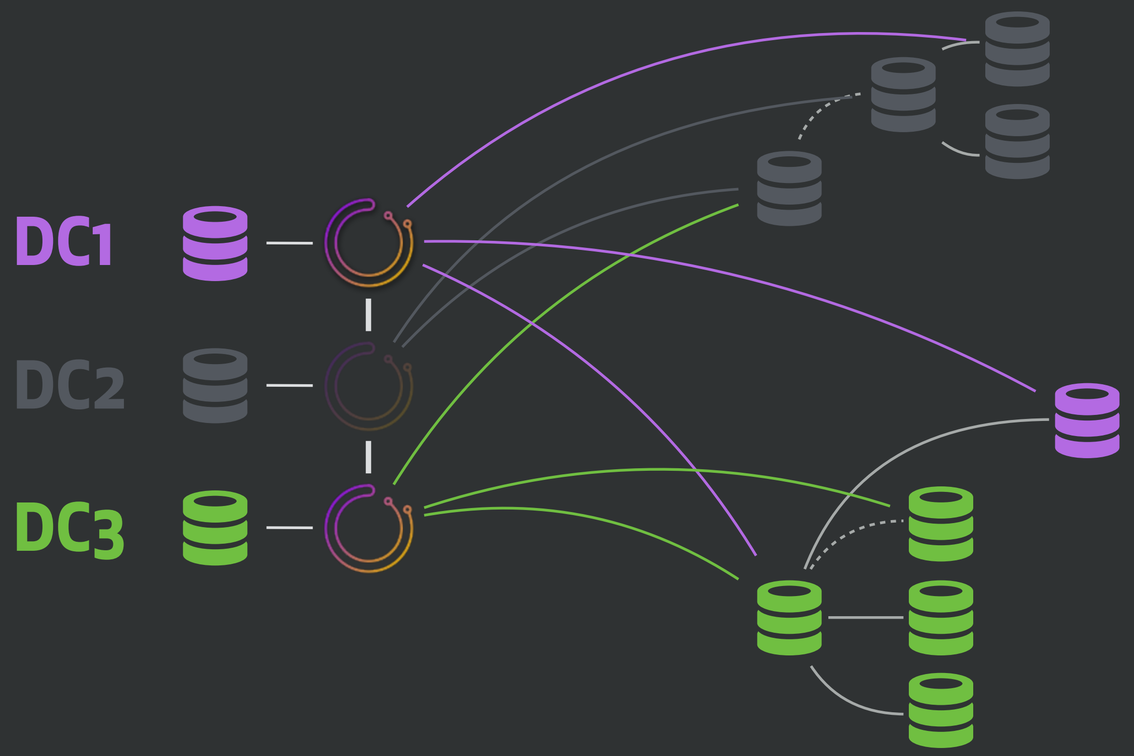
- From the point of view of DC2 servers, and in particular in the point of view of DC2's orchestrator node:
  - Master and replicas are fine.
  - DC1 and DC3 servers are all dead.
  - No need for fail over.
- However, DC2's orchestrator is **not part of a quorum**, hence not the leader. **It doesn't call the shots**.

orchestrator/raft: fencing



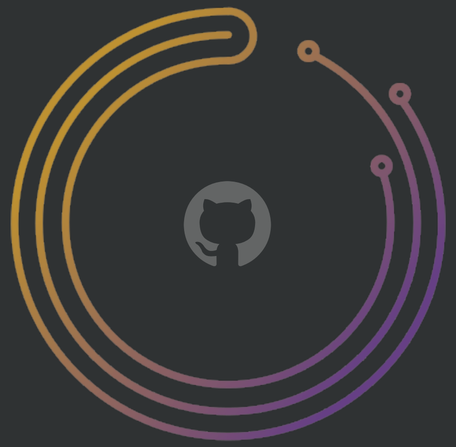
- In the eyes of either DC1's or DC3's orchestrator:
  - All DC2 servers, including the master, are dead.
  - There is need for failover.
- DC1's and DC3's orchestrator nodes form a quorum. One of them will become the leader.
- The leader will initiate failover.

orchestrator/raft: fencing



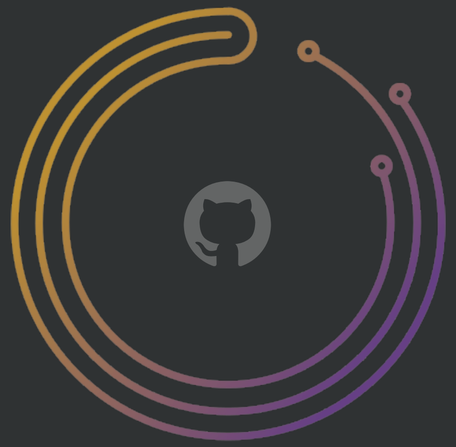
- Depicted potential failover result. New master is from DC3.
- The topology is detached and split into two.
- orchestrator nodes will keep attempting to contact DC2 servers.
- When DC2 is back:
  - DC2 MySQL nodes still identified as “broken”
  - DC2’s orchestrator will rejoin the quorum, and catch up with the news.

HAProxy setup



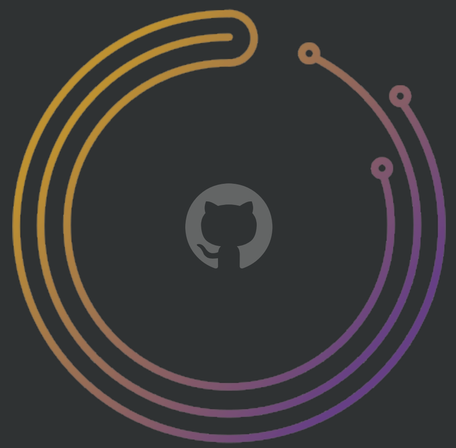
```
listen orchestrator
  bind  0.0.0.0:80 process 1
  bind  0.0.0.0:80 process 2
  bind  0.0.0.0:80 process 3
  bind  0.0.0.0:80 process 4
  mode tcp
  option httpchk GET /api/leader-check
  maxconn 20000
  balance first
  retries 1
  timeout connect 1000
  timeout check 300
  timeout server 30s
  timeout client 30s

  default-server port 3000 fall 1 inter 1000 rise 1 downinter 1000 on-
marked-down shutdown-sessions weight 10

  server orchestrator-node-0 orchestrator-node-0.fqdn.com:3000 check
  server orchestrator-node-1 orchestrator-node-1.fqdn.com:3000 check
  server orchestrator-node-2 orchestrator-node-2.fqdn.com:3000 check
```

orchestrator-client setup



- Create and edit **/etc/profile.d/orchestrator-client.sh**
- if exists, **orchestrator-client** inlines this file.
- Choose:
  - List all orchestrator nodes
    - **orchestrator-client** will iterate in real time to detect the leader. No proxy needed.
  - Proxy node(s)

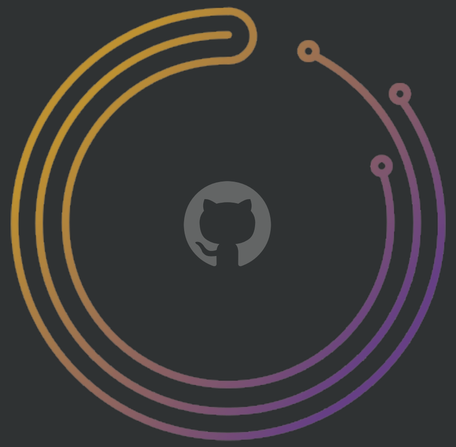
```
export ORCHESTRATOR_API="https://orchestrator.host1:3000/api https://
orchestrator.host2:3000/api https://orchestrator.host3:3000/api"

export ORCHESTRATOR_API="https://orchestrator.proxy:80/api"
```



Security

- Control access to orchestrator
- Support read-only mode
- Basic auth
- Headers authentication via proxy

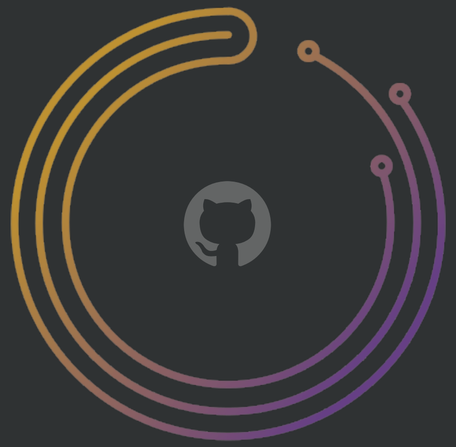
Security: none



- Everyone can read
- Everyone can operate (relocate replicas, stop/start replication, set read-only, RESET SLAVE ALL)
- Everyone is all-powerful

```
{
  "AuthenticationMethod": "",
}
```

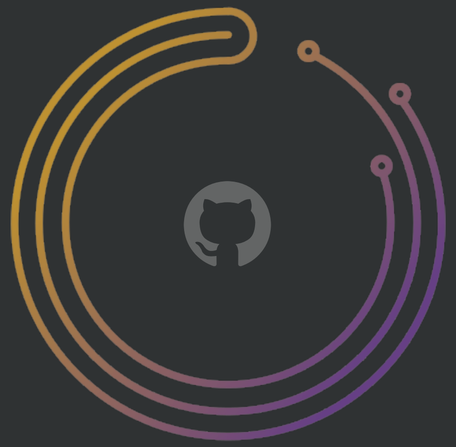
Security: read-only



- Everyone can read
- No one can operate

```
{
  "ReadOnly": true,
}
```

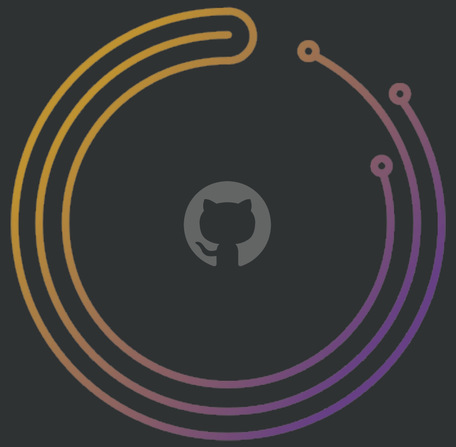
Security: basic



- Basic Auth: a simple HTTP authentication protocol
- User/password
- No login/logout
- All-powerful

```
{
  "AuthenticationMethod": "basic",
  "HTTPAuthUser":         "dba_team",
  "HTTPAuthPassword":     "time_for_dinner",
}
```

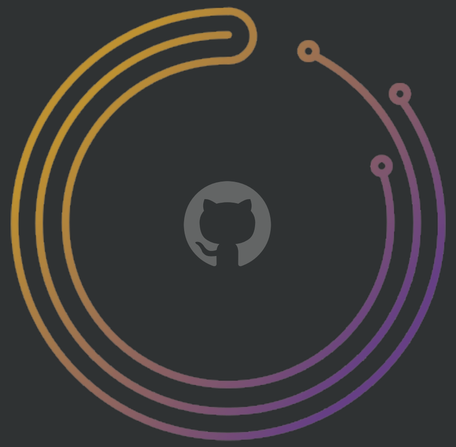
Security: multi



- Extends basic auth
- Either provide credentials
  - makes you all-powerful
- Or use “read-only” as username, whatever password
  - gets you read-only access

```
{
  "AuthenticationMethod": "multi",
  "HTTPAuthUser":         "dba_team",
  "HTTPAuthPassword":     "time_for_dinner",
}
```

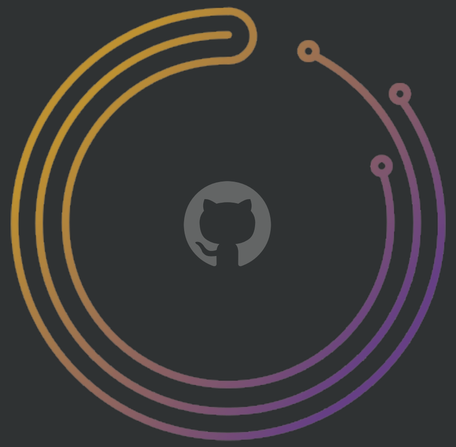
Security: headers



- Put your favorite proxy in front of orchestrator
  - Apache, nginx, …
- Bind to local, no external connections
- Expect proxy to provide user via header
- **PowerAuthUsers** are all-powerful. The rest are read-only

```
{
  "ListenAddress": "127.0.0.1:3000",
  "AuthenticationMethod": "proxy",
  "AuthUserHeader": "X-Forwarded-User",
  "PowerAuthUsers": [
    "wallace", "gromit", "shaun"
  ],
}
```

Security: headers

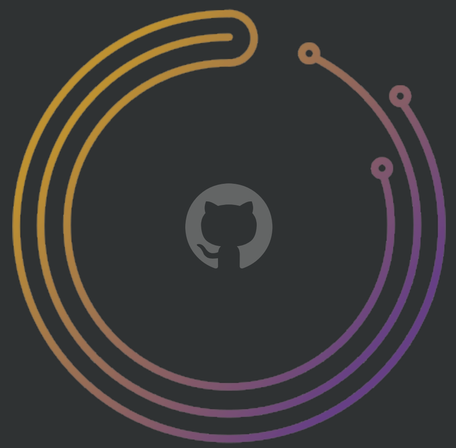


- A apache2 setup may look like this.
- Integrate with LDAP

```
RequestHeader unset X-Forwarded-User
RewriteEngine On
RewriteCond %{LA-U:REMOTE_USER} (.+)
RewriteRule .* - [E=RU:%1,NS]
RequestHeader set X-Forwarded-User %{RU}e
```

Agenda

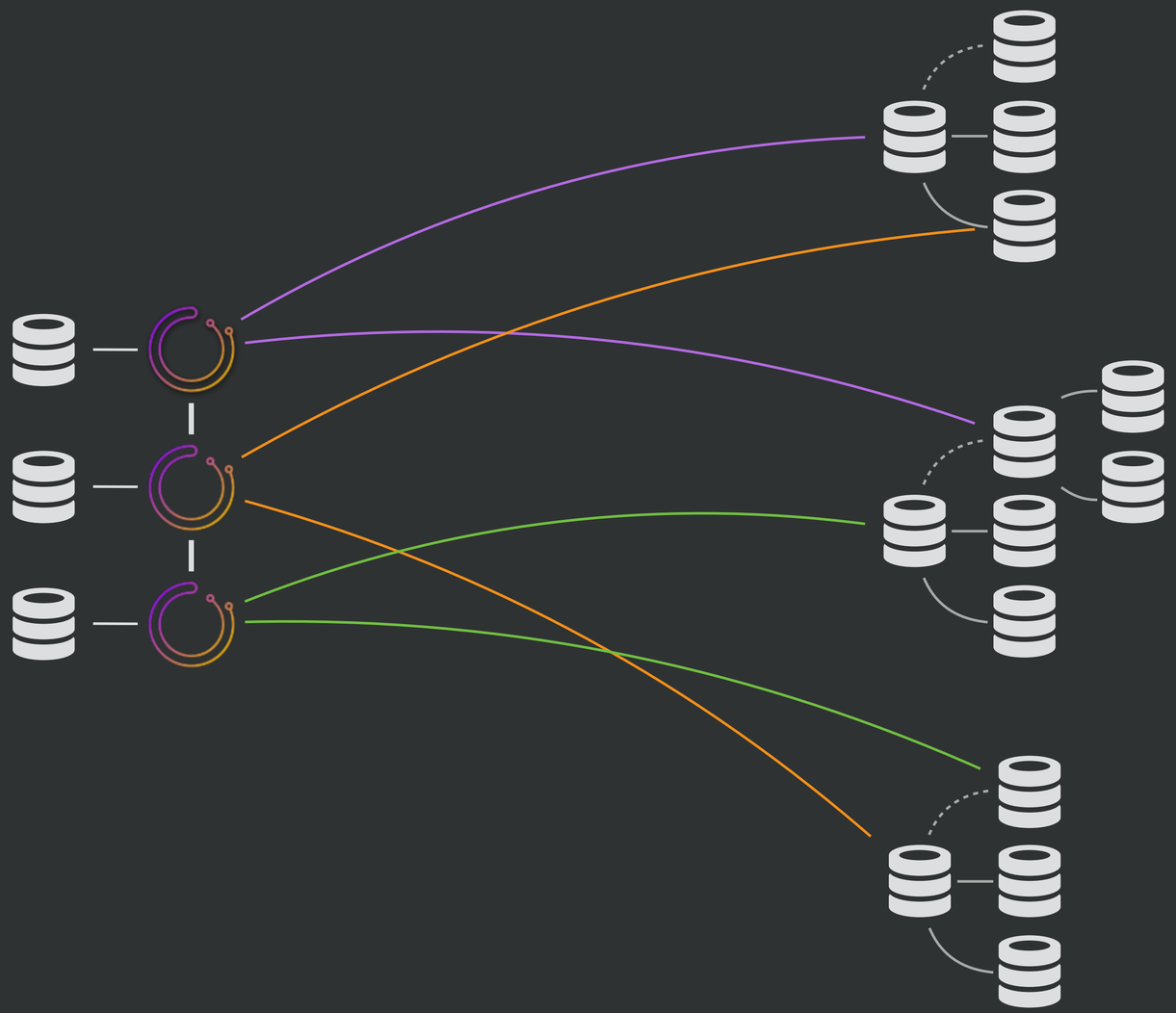
- Setting up orchestrator
  - Backend
  - Discovery
  - Refactoring
  - Detection & recovery
- Scripting
- HA
  - Raft cluster
- Deployment
- Roadmap



Roadmap

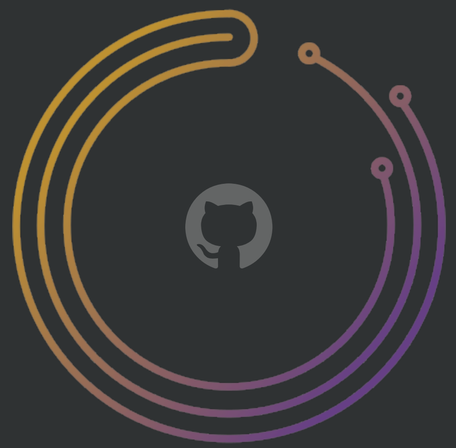
- orchestrator/raft: dynamic node join/leave
- Distributed probing
- The Great Configuration Variables Exodus
  - Simplifying config, continued work
- Thoughts on integrations
  - Consul/proxy

Roadmap: distributed probing



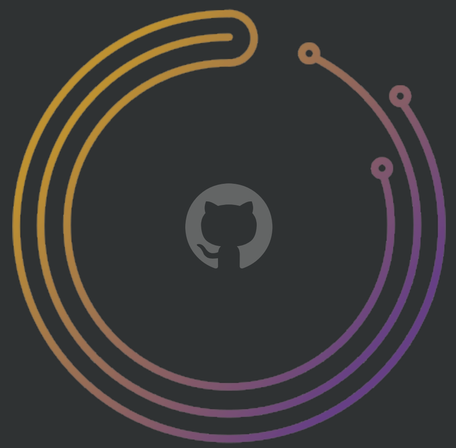
- Leader distributes probing across available (healthy) nodes
- Applies to both shared backend DB and raft setups

Supported setups



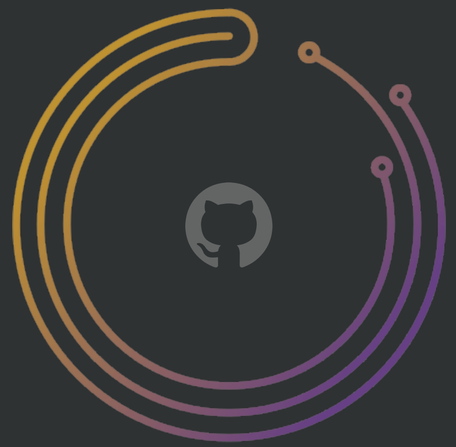
- “Classic” replication
- GTID (Oracle, MariaDB)
- Master-Master
- Semi-sync
- STATEMENT, MIXED, ROW
- Binlog servers
- Mixture of all the above, mixtures of versions

Unsupported setups



- Galera
  - TODO? *possibly*
- InnoDB Cluster
  - TODO? *possibly*
- Multisource
  - TODO? *probably not*
- Tungsten
  - TODO? *no*

GitHub talks



- gh-ost: triggerless, painless, trusted online schema migrations

  Jonah Berquist, Wednesday 27 September , 14:20

  https://www.percona.com/live/e17/sessions/gh-ost-triggerless-painless-trusted-online-schema-migrations
- MySQL Infrastructure Testing Automation at GitHub

  Tom Krouper, Shlomi Noach, Wednesday 27 September , 15:20

  https://www.percona.com/live/e17/sessions/mysql-infrastructure-testing-automation-at-github

orchestrator talks

- Rolling out Database-as-a-Service using ProxySQL and Orchestrator

  Matthias Crauwels (Pythian), Tuesday 26 September , 15:20

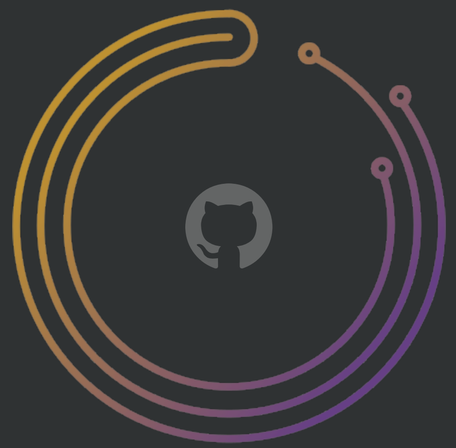
  https://www.percona.com/live/e17/sessions/rolling-out-database-as-a-service-using-proxysql-and-orchestrator
- Orchestrating ProxySQL with Orchestrator and Consul

  Avraham Apelbaum (Wix.COM), Wednesday 27 September , 12:20

  https://www.percona.com/live/e17/sessions/orchestrating-proxysql-with-orchestrator-and-consul

Thank you!

Questions?



github.com/shlomi-noach
@ShlomiNoach