



Practical Orchestrator

Shlomi Noach

GitHub

Percona Live 2017

Agenda



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

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



GitHub

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**

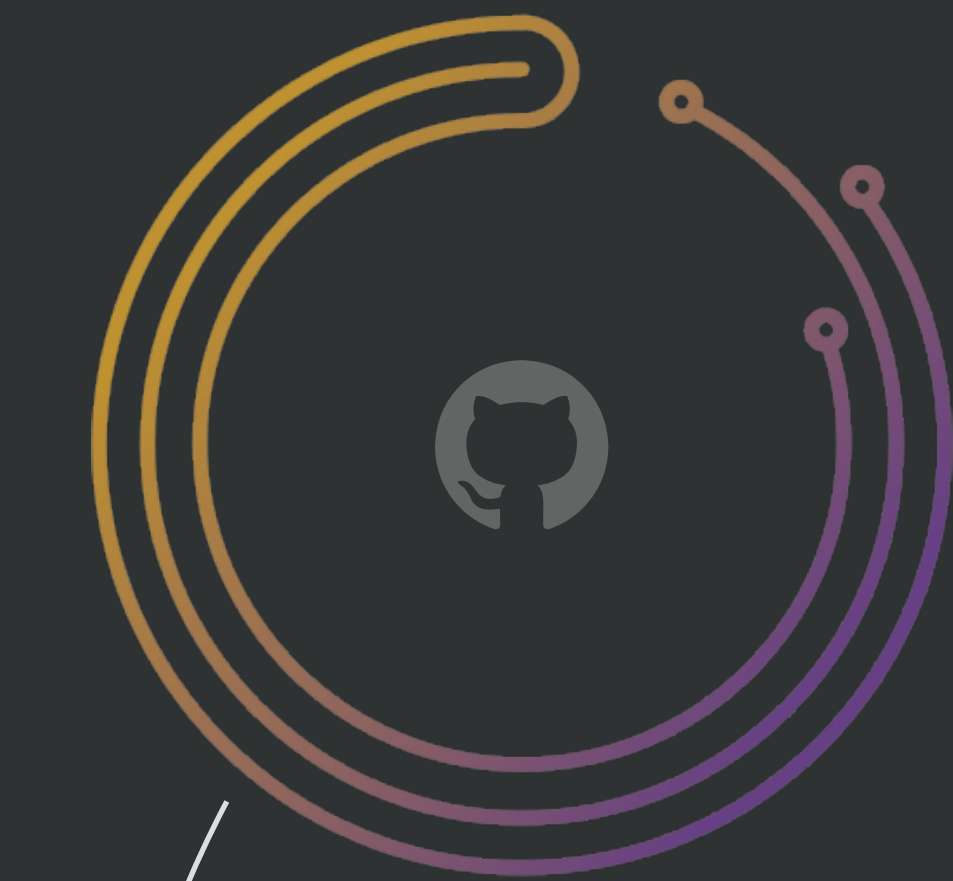
Relocate replicas, manipulate, detach, reorganize

- **Recovery**

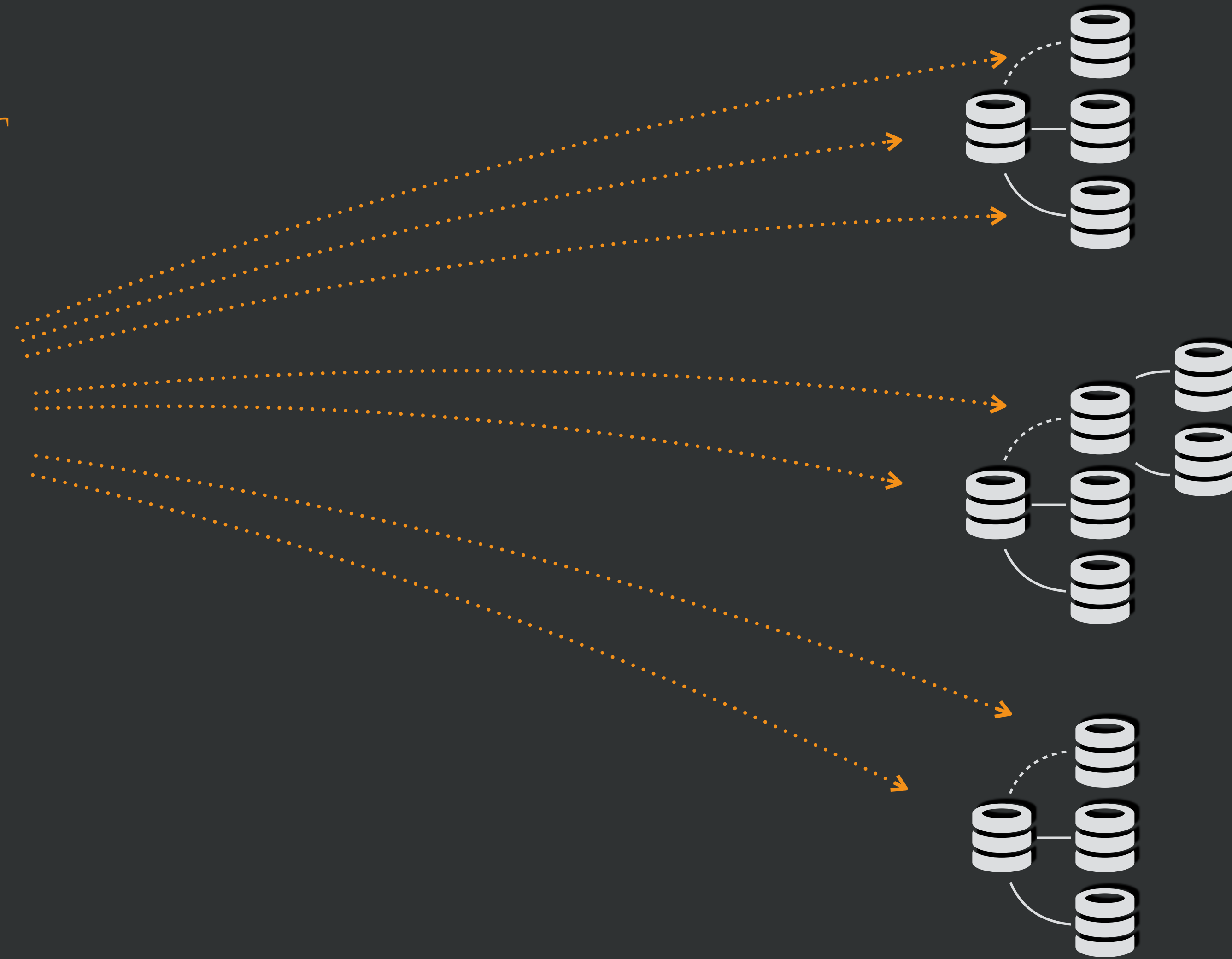
Analyze, detect crash scenarios, structure warnings, failovers, promotions, acknowledgements, flap control, downtime, hooks

Deployment in a nutshell

orchestrator



backend DB



Agenda



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

Basic & backend setup



- Let orchestrator know where to find backend database
- 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 backend



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

Agenda



- Setting up orchestrator
 - Backend
 - **Discovery**
 - Refactoring
 - Detection & recovery
- Scripting
- HA
- 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
- Updating a meta table to learn about Pseudo-GTID updates.
- <https://github.com/github/orchestrator/tree/master/resources/pseudo-gtid>

```
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)
```

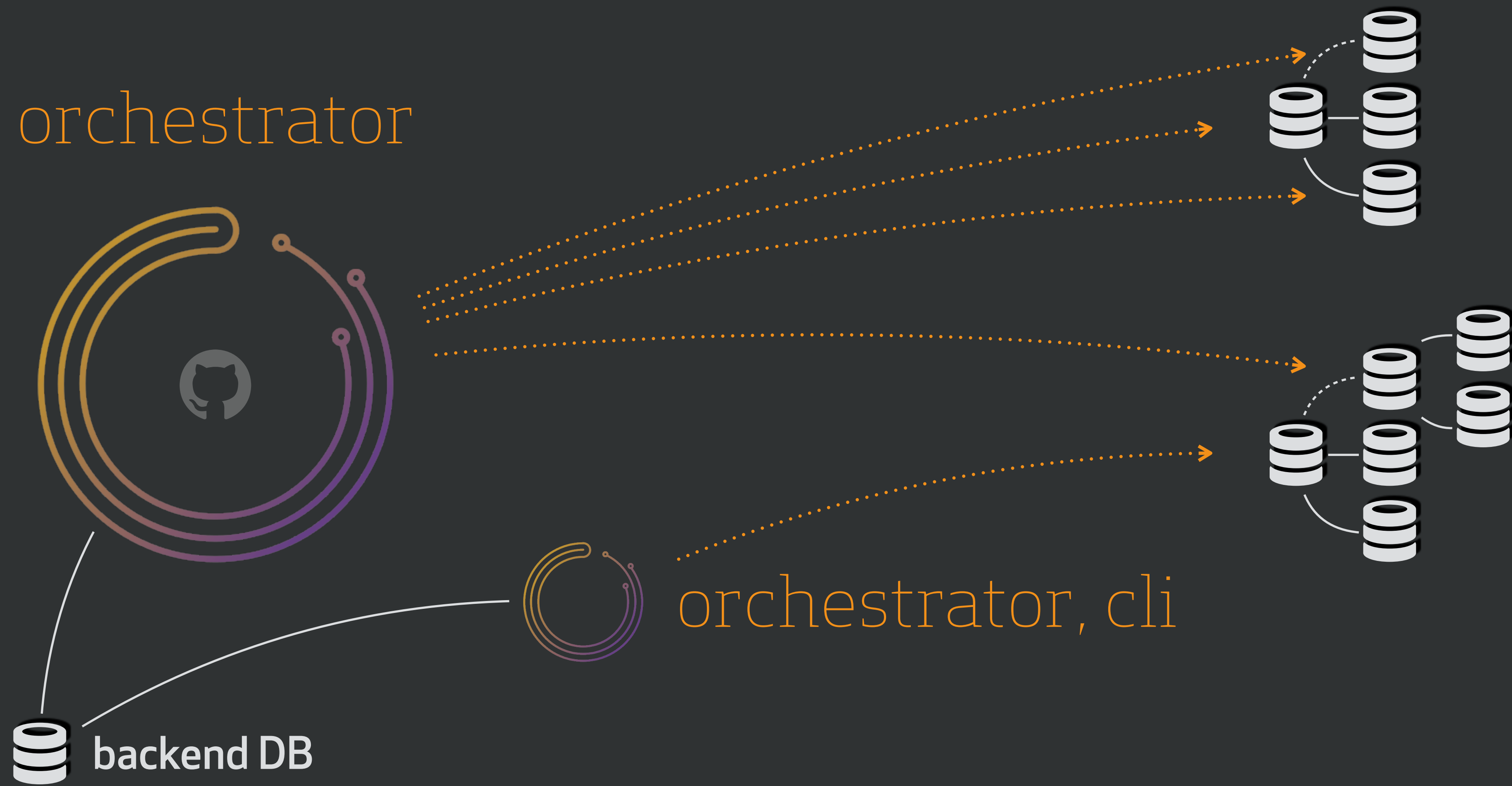
Pseudo-GTID



- 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",  
}
```

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

CLI: information



```
orchestrator -c clusters
```

```
orchestrator -c all-instances
```

```
orchestrator -c which-cluster some.instance.in.cluster
```

```
orchestrator -c which-cluster-instances -alias mycluster
```

```
orchestrator -c which-master some.instance
```

```
orchestrator -c which-replicas some.instance
```

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

Agenda



- Setting up orchestrator
 - Backend
 - Discovery
 - **Refactoring**
 - Detection & recovery
- Scripting
- HA
- Roadmap

CLI: refactoring



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

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

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

CLI: refactoring



- file:pos specific

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

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

CLI: various commands



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

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

```
orchestrator -c set-writeable -i some.instance.com
```

```
orchestrator -c stop-slave -i some.instance.com
```

```
orchestrator -c start-slave -i some.instance.com
```

```
orchestrator -c restart-slave -i some.instance.com
```

```
orchestrator -c skip-query -i some.instance.com
```

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

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

CLI: some fun



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

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

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

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

API



- The web interface is merely a facade for API calls
- Anything done from CLI can be done from API

```
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
- Roadmap

Recovery: basic config



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

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

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": [  
    "*"   
  ],  
}
```

Recovery, CLI



```
orchestrator -c replication-analysis
```

```
orchestrator -c recover -i a.dead.instance.com
```

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

```
orchestrator -c graceful-master-takeover -alias mycluster
```

```
orchestrator -c force-master-takeover -i replica.to.forcefully.promote # danger zone
```

```
orchestrator -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
- Roadmap

Scripting: master failover testing automation



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

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

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

```
intermediate_master=$(orchestrator -c which-replicas -i $master | shuf | head -1)  
orchestrator -c which-replicas -i $master | grep -v $intermediate_master | shuf |  
head -2 | while read i ; do \  
    orchestrator -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=$(orchestrator -c which-cluster-master -alias mycluster)
[ -z "$new_master" ] && { echo "strange, cannot find master" ; exit 1 ; }
[ "$new_master" == "$master" ] && { echo "no change of master" ; exit 1 ; }
orchestrator -c which-cluster-instances -alias mycluster | while read i ; do \
    orchestrator -c relocate -i $i -d $new_master \
done
count_replicas=$(orchestrator -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`

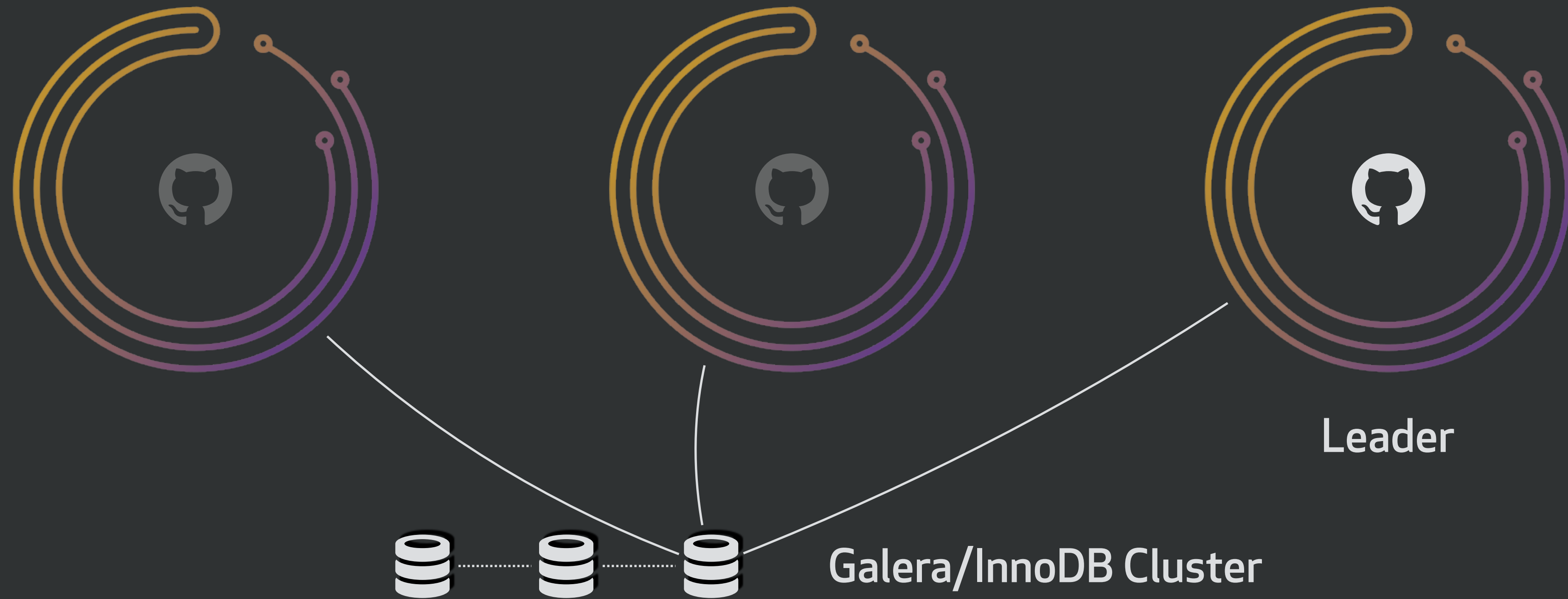
Agenda



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

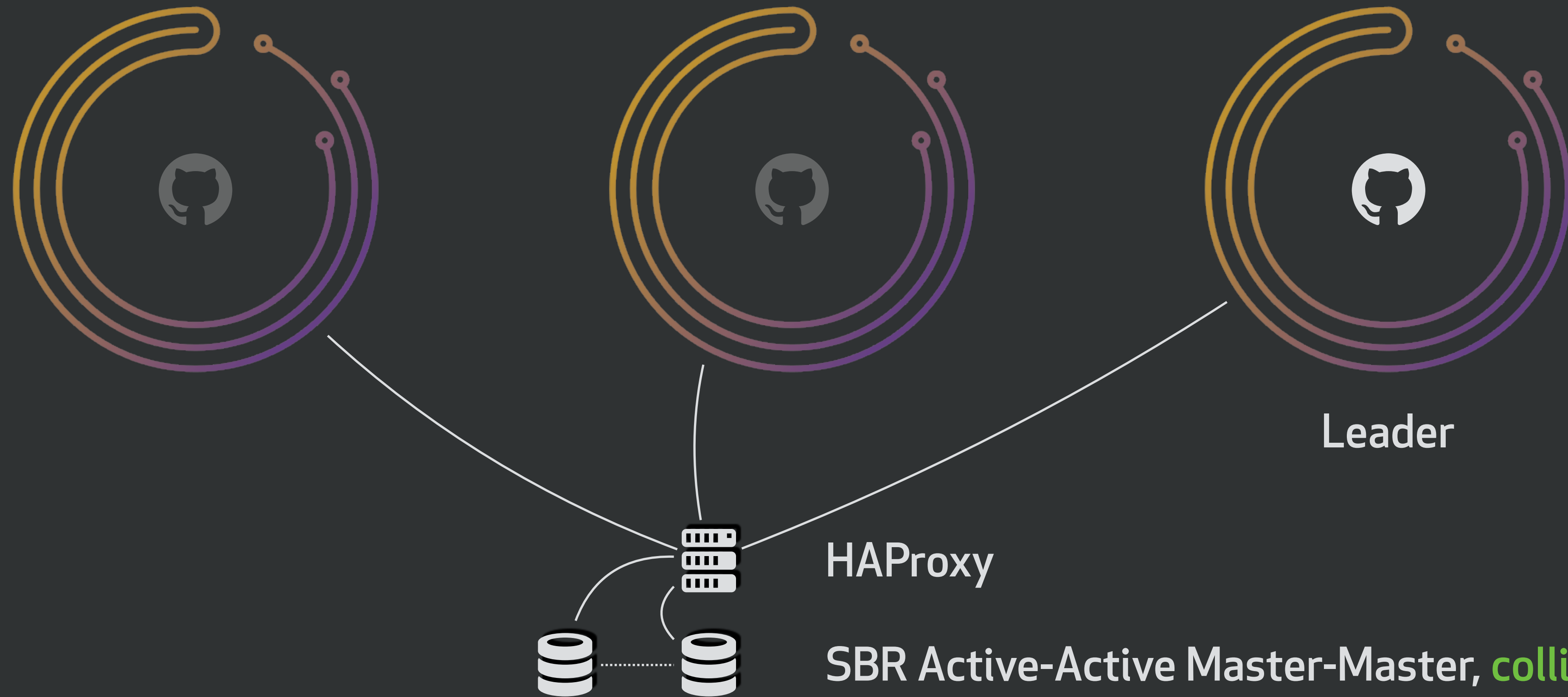
HA

orchestrator



HA

orchestrator



HA: on the roadmap

orchestrator



Each orchestrator node with a local DB, MySQL/SQLite

Raft consensus for leadership and events changelog

Agenda



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

Roadmap



- SQLite backend (existing POC)
- Raft consensus
- Improving GTID support
- The Great Configuration Variables Exodus
 - Simplifying config
- Thoughts on integrations

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, Tuesday 25 April , 14:20

<https://www.percona.com/live/17/sessions/gh-ost-triggerless-painless-trusted-online-schema-migrations>

- **Automating Schema Changes using gh-ost**

Tom Krouper, Thursday 27 April, 12:50

<https://www.percona.com/live/17/sessions/automating-schema-changes-using-gh-ost>

- **Practical JSON in MySQL 5.7 and beyond**

Ike Walker, Thursday 27 April, 15:00

<https://www.percona.com/live/17/sessions/practical-json-mysql-57-and-beyond>

Thank you!

Questions?



github.com/shlomi-noach
[@ShlomiNoach](https://twitter.com/ShlomiNoach)