Autopsy of a MySQL automation disaster

by Jean-François Gagné
presented at SRECon Dublin (October 2019)
Senior Infrastructure Engineer / System and MySQL Expert
jeanfrancois AT messagebird DOT com / @jfg956 #SREcon

To err is human

To really foul things up requires a computer[1]

(or a script)

[1]: http://quoteinvestigator.com/2010/12/07/foul-computer/

Session Summary

1. MySQL replication
2. Automation disaster: external eye
3. Chain of events: analysis
4. Learning / takeaway

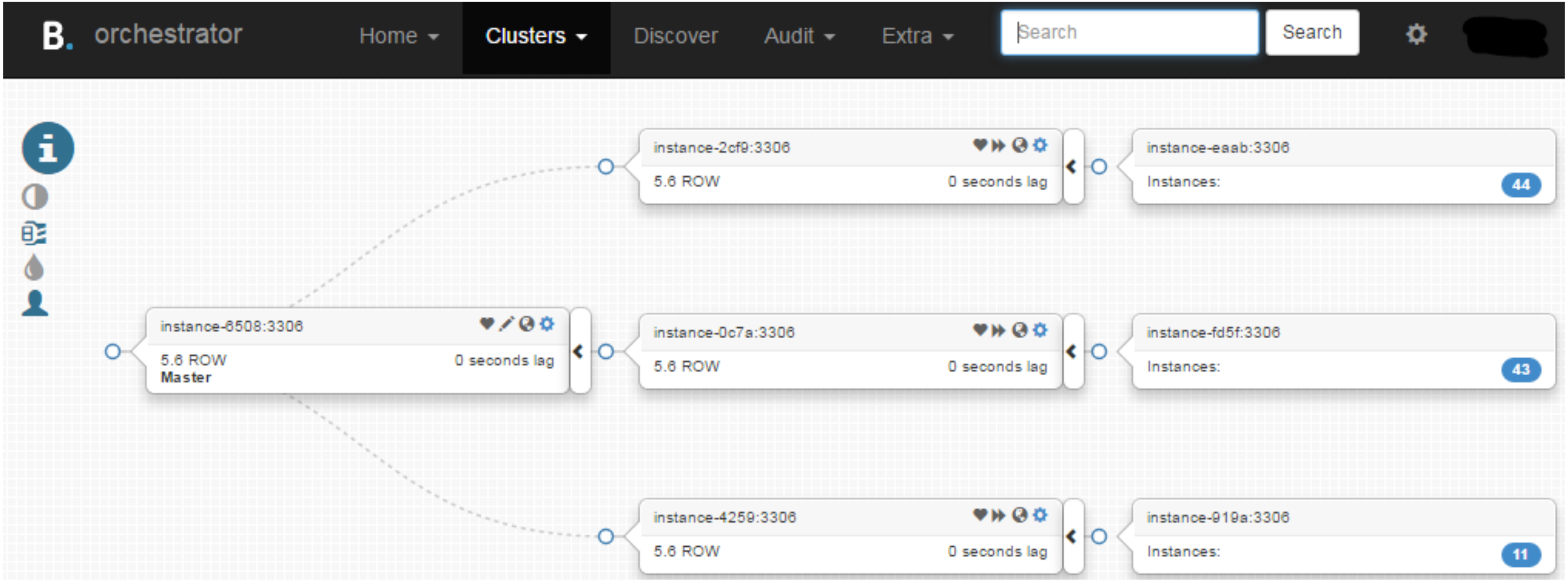
MySQL replication

- Typical MySQL replication deployment at Booking.com:

```
+----+
| M  |
+----+
  |
  +------+-- ... --+---------------+-------- ...
  |      |         |               |
+----+ +----+    +----+          +----+
| S1 | | S2 |    | Sn |          | M1 |
+----+ +----+    +----+          +----+
                                   |
                                   +-- ... --+
                                   |         |
                                 +----+    +----+
                                 | T1 |    | Tm |
                                 +----+    +----+
```

MySQL replication’

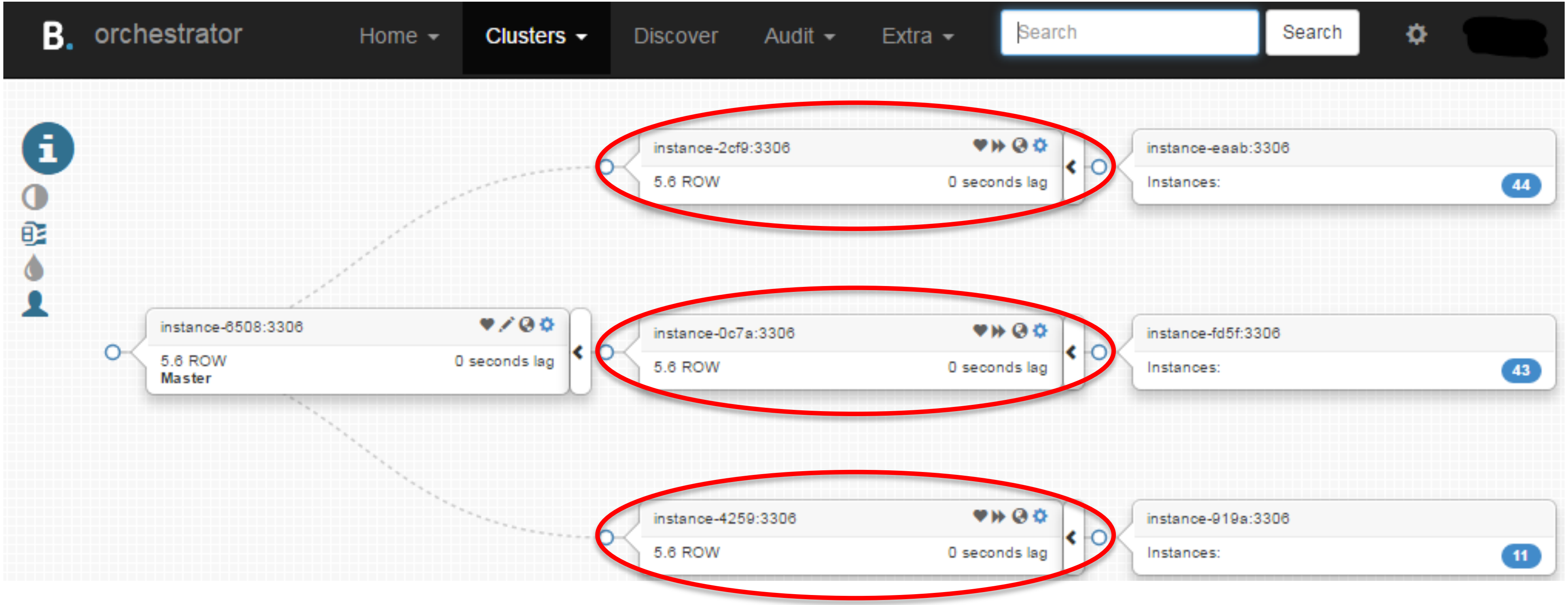
- And they use(d) Orchestrator (more about Orchestrator in the next slide):



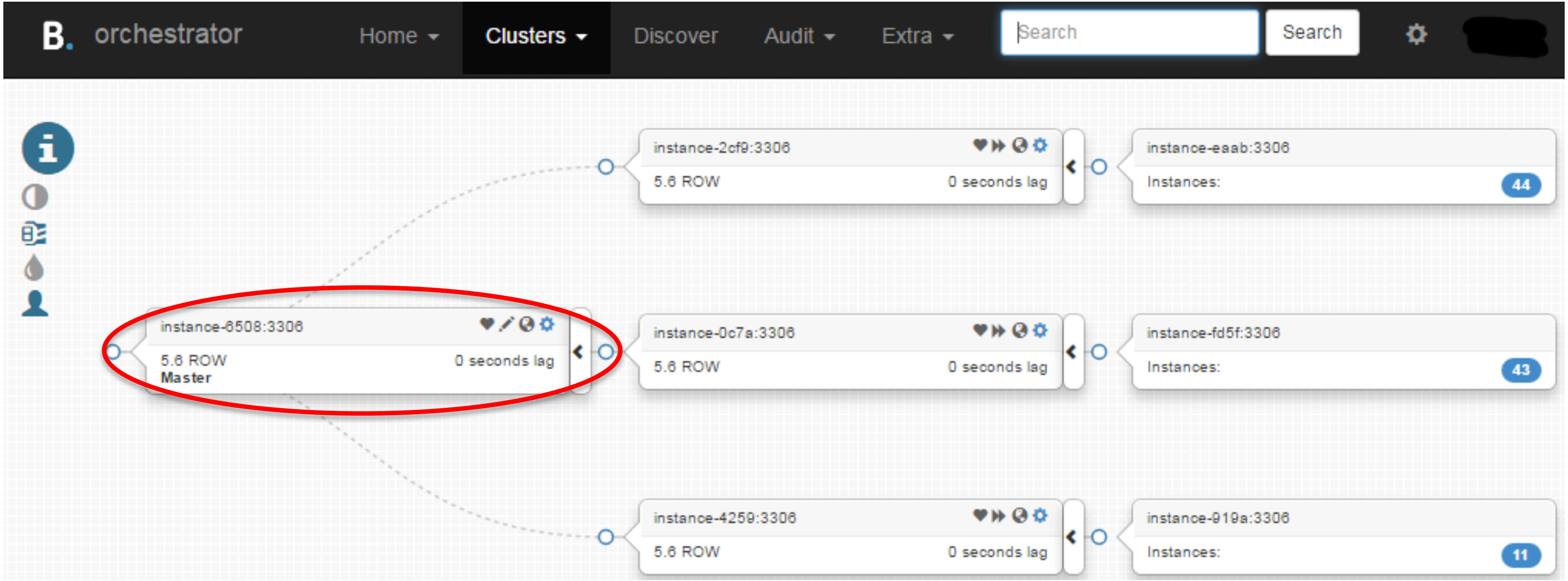
MySQL replication”

- Orchestrator allows to:
  - Visualize replication deployments
  - Move slaves for planned maintenance of an intermediate master
  - Automatically replace an intermediate master in case of its unexpected failure (thanks to pseudo-GTIDs when we have not deployed GTIDs)
  - Automatically replace a master in case of a failure (failing over to a slave)
- But Orchestrator cannot replace a master alone:
  - Booking.com uses DNS for master discovery
  - So Orchestrator calls a homemade script to repoint DNS (and to do other magic)

Intermediate Master Failure



Master Failure



MySQL Master High Availability [1 of 4]

Failing-over the master to a slave is my favorite HA method

- But it is not as easy as it sounds, and it is hard to automate <u>well</u>
- An example of complete failover solution in production:
  https://github.blog/2018-06-20-mysql-high-availability-at-github/

The five considerations of master high availability:
(https://jfg-mysql.blogspot.com/2019/02/mysql-master-high-availability-and-failover-more-thoughts.html)

- <u>Plan</u> how you are doing master high availability
- Decide <u>when</u> you apply your plan (Failure Detection – FD)
- <u>Tell</u> the application about the change (Service Discovery – SD)
- <u>Protect</u> against the limit of FD and SD for avoiding split-brains (Fencing)
- <u>Fix</u> your data if something goes wrong

MySQL Master High Availability [2 of 4]

Failure detection (*FD*) is the 1st part (and 1st challenge) of failover

- It is a very hard problem: partial failure, unreliable network, partitions, …
- It is impossible to be 100% sure of a failure, and confidence needs time
  → quick FD is unreliable, relatively reliable FD implies longer unavailability

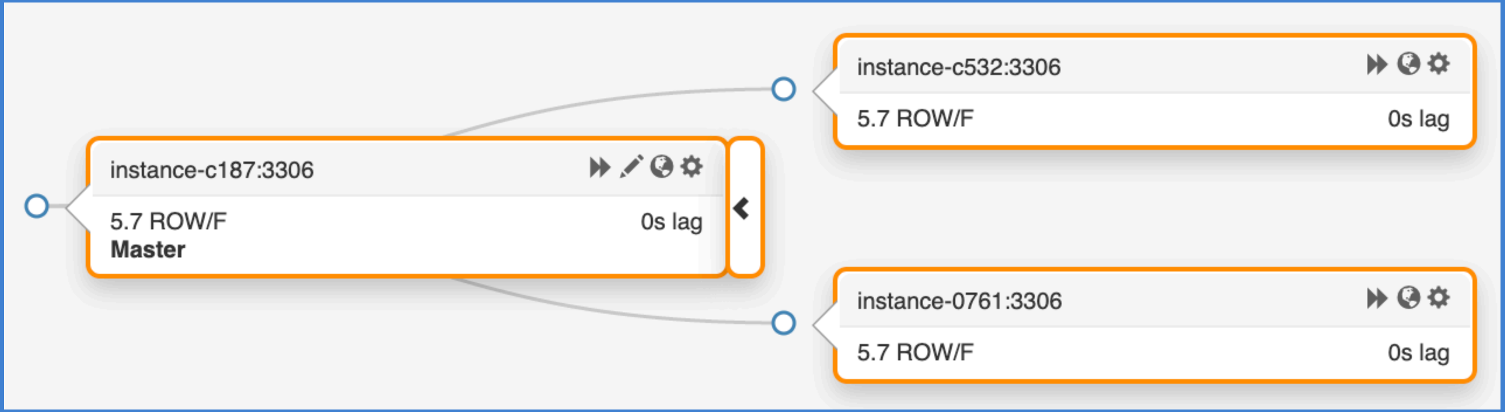
➢ You need to accept that FD generates false positive (and/or false negative)

Repointing is the 2nd part of failover:

- Relatively easy with the right tools: MHA, GTID, Pseudo-GTID, Binlog Servers, …
- Complexity grows with the number of direct slaves of the master
  (what if you cannot contact some of those slaves…)
- Some software for repointing:
  - Orchestrator, Ripple Binlog Server,
    Replication Manager, MHA, Cluster Control, MaxScale, …

MySQL Master High Availability [3 of 4]

In this configuration and when the master fails,
one of the slave needs to be repointed to the new master:



MySQL Master High Availability [4 of 4]

Service Discovery (*SD*) is the 3rd part (and 2nd challenge) of failover:

- If centralised, it is a SPOF; if distributed, impossible to update atomically
- SD will either introduce a bottleneck (including performance limits) or will be unreliable in some way (pointing to the wrong master)
- Some ways to implement MySQL Master SD: DNS, ViP, Proxy, Zookeeper, …
  http://code.openark.org/blog/mysql/mysql-master-discovery-methods-part-1-dns

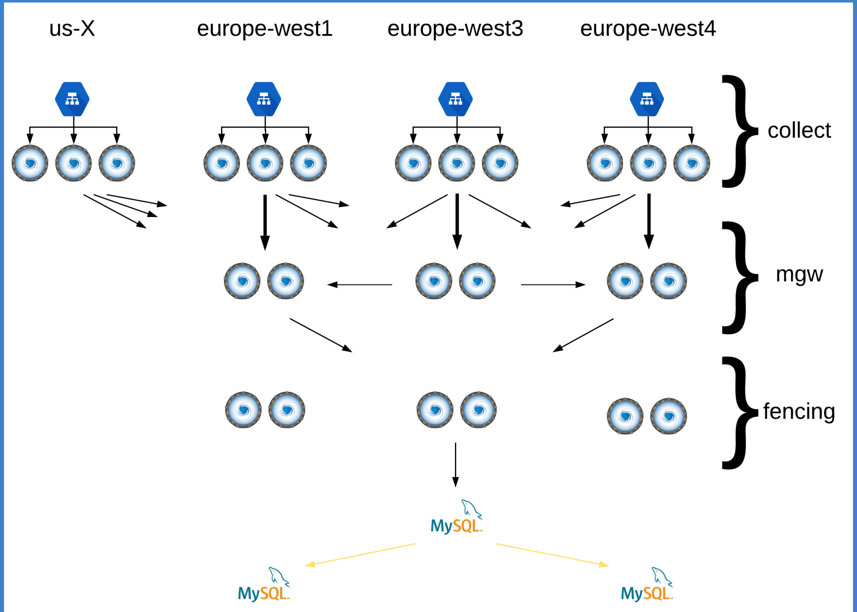
➢ Unreliable FD and unreliable SD is a recipe for split-brains !

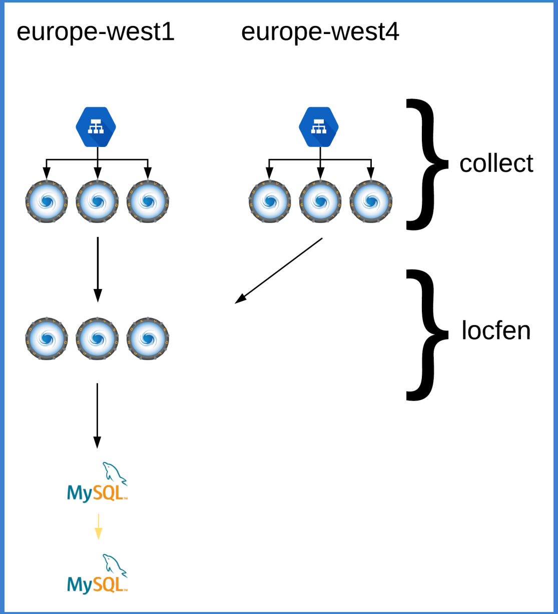
Protecting against split-brains (Fencing): Adv. Subject – not many solutions
(Proxies and semi-synchronous replication might help)

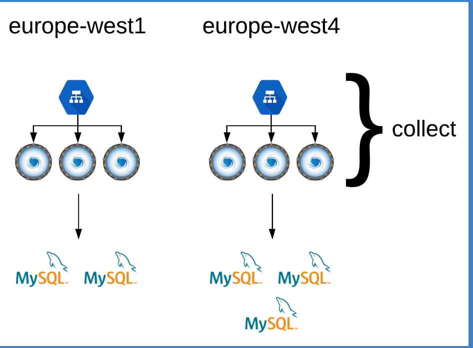
Fixing your data in case of a split-brain: only you can know how to do this !
(tip on this later in the talk)

MySQL Service Discovery @ MessageBird

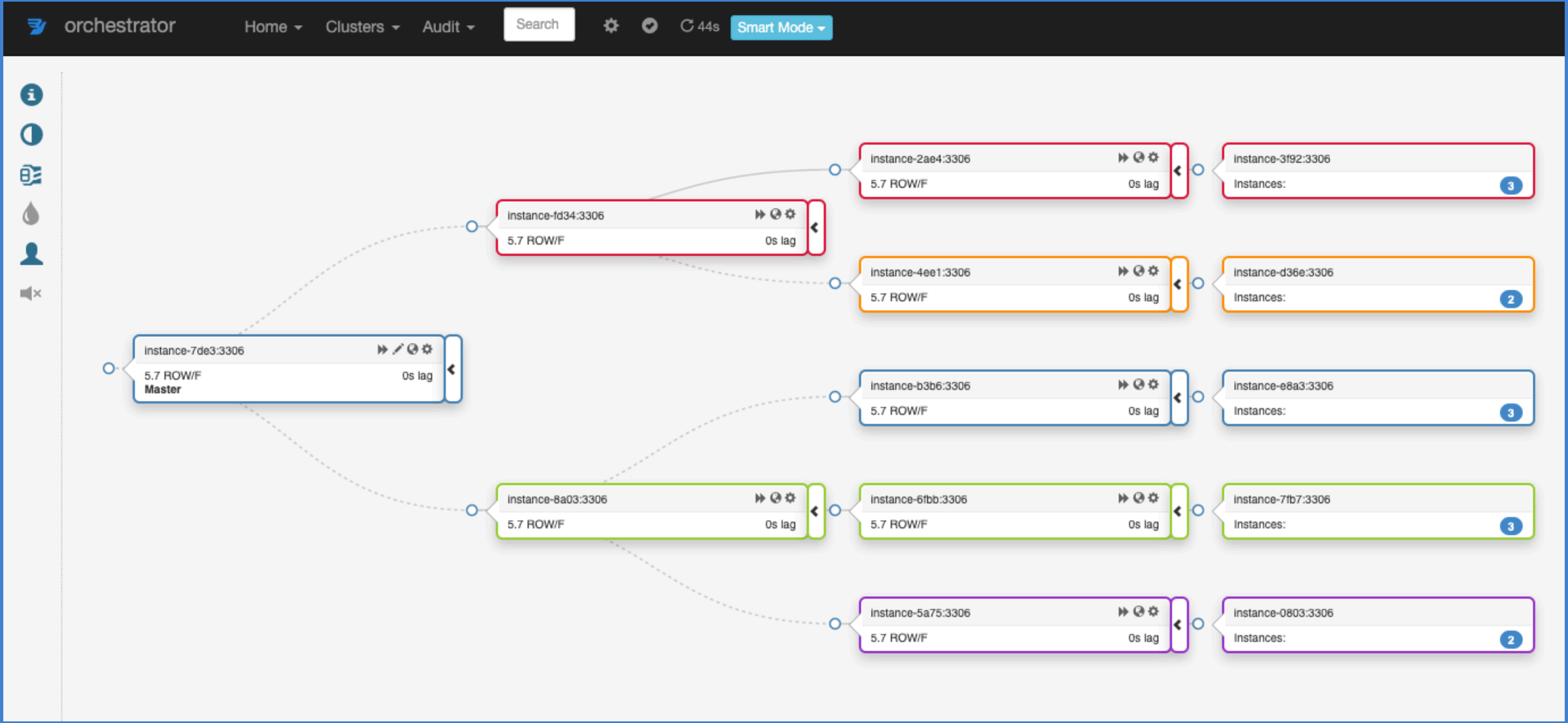
MessageBird uses ProxySQL for MySQL Service Discovery







Orchestrator @ MessageBird



Failover War Story

Master failover does not always go as planned

We will now look at our War Story

Our subject database

- Simple replication deployment (in two data centers):

```
Master RWs         +----+
happen here -->    | A  |
                   +----+
                     |
                     +------------------------+
                     |                        |
Reads              +----+                  +----+
happen here -->    | B  |                  | X  |
                   +----+                  +----+
                                              |
                                           +----+      And reads
                                           | Y  |  <-- happen here
                                           +----+
```

Incident: 1^{st} event

- A and B (two servers in same data center) fail at the same time:

```
Master RWs         +\-/+
happen here -->    | A |
but now failing    +/-\+


Reads              +\-/+                            +----+
happen here -->    | B |                            | X  |
but now failing    +/-\+                            +----+
                                                       |
                                                    +----+     Reads
                                                    | Y  | <-- happen here
                                                    +----+
```

(I will cover how/why this happened later.)

Incident: 1st event'

- Orchestrator fixes things:

```
                 +\-/+
                 | A |
                 +/-\+



Reads            +\-/+                       +----+      Now, Master RWs
happen here -->  | B |                       | X  |  <-- happen here
but now failing  +/-\+                       +----+
                                               |
                                             +----+      Reads
                                             | Y  |  <-- happen here
                                             +----+
```

Split-brain: disaster

- A few things happen in that day and night, and I wake-up to this:

```
                  +\-/+
                  | A |
                  +/-\+



 Master RWs       +---+                   +---+
 happen here -->  | B |                   | X |
                  +---+                   +---+
                                            |
                                          +---+
                                          | Y |
                                          +---+
```

Split-brain: disaster’

- And to make things worse, reads are still happening on Y:

```
               +\-/+
               | A |
               +/-\+


  Master RWs   +---+               +---+
happen here -->| B |               | X |
               +---+               +---+
                                     |
                                   +---+      Reads
                                   | Y | <--  happen here
                                   +---+
```

Split-brain: disaster”

- This is not good:
  - When A and B failed, X was promoted as the new master
  - Something made DNS point to B (we will see what later)
    → writes are now happening on B
  - But B is outdated: all writes to X (after the failure of A) did not reach B
  - So we have data on X that cannot be read on B
  - And we have new data on B
    that is not read on Y

```
                    +\-/+
                    | A |
                    +/-\+

Master RWs          +---+     +---+
happen here  -->    | B |     | X |
                    +---+     +---+
                                |
                              +---+      Reads
                              | Y | <--  happen here
                              +---+
```

Split-brain: analysis

- Digging more in the chain of events, we find that:
  - After the 1^{st} failure of A, a 2^{nd} one was detected and Orchestrator failed over to B
  - So after their failures, A and B came back and formed an isolated replication chain

```
+\-/+
| A |
+/-\+

+\-/+     +---+      Master RWs
| B |     | X | <-- happen here
+/-\+     +---+
            |
          +---+      Reads
          | Y | <-- happen here
          +---+
```

Split-brain: analysis

- Digging more in the chain of events, we find that:
  - After the 1st failure of A, a 2nd one was detected and Orchestrator failed over to B
  - So after their failures, A and B came back and formed an isolated replication chain
  - And something caused a failure of A

```
+---+
| A |
+---+
  |
+---+     +---+      Master RWs
| B |     | X | <--  happen here
+---+     +---+
            |
          +---+      Reads
          | Y | <--  happen here
          +---+
```

Split-brain: analysis

- Digging more in the chain of events, we find that:
  - After the 1st failure of A, a 2nd one was detected and Orchestrator failed over to B
  - So after their failures, A and B came back and formed an isolated replication chain
  - And something caused a failure of A
- But how did DNS end-up pointing to B ?
  - The failover to B called the DNS repointing script
  - The script stole the DNS entry from X and pointed it to B

```
+\-/+
| A |
+/-\+

+---+      +---+      Master RWs
| B |      | X | <--  happen here
+---+      +---+
             |
           +---+      Reads
           | Y | <--  happen here
           +---+
```

Split-brain: analysis

- Digging more in the chain of events, we find that:
  - After the 1^{st} failure of A, a 2^{nd} one was detected and Orchestrator failed over to B
  - So after their failures, A and B came back and formed an isolated replication chain
  - And something caused a failure of A
- But how did DNS end-up pointing to B ?
  - The failover to B called the DNS repointing script
  - The script stole the DNS entry from X and pointed it to B
  - But is that all: what made A fail ?

```
                 +\-/+
                 | A |
                 +/-\+

Master RWs       +----+     +----+
happen here -->  | B  |     | X  |
                 +----+     +----+
                              |
                            +----+     Reads
                            | Y  | <-- happen here
                            +----+
```

Split-brain: analysis’

- What made A fail ?
  - Once A and B came back up as a new replication chain, they had outdated data
  - If B would have come back before A, it could have been re-slaved to X

```
+----+
| A  |
+----+
  |
+----+     +----+     Master RWs
| B  |     | X  | <-- happen here
+----+     +----+
             |
           +----+     Reads
           | Y  | <-- happen here
           +----+
```

Split-brain: analysis’

- What made A fail ?
  - Once A and B came back up as a new replication chain, they had outdated data
  - If B would have come back before A, it could have been re-slaved to X
  - But because A came back before re-slaving, it injected heartbeat and p-GTID to B

```
+\-/+
| A |
+/-\+

+----+     +----+      Master RWs
| B |      | X | <-- happen here
+----+     +----+
             |
           +----+      Reads
           | Y | <-- happen here
           +----+
```

Split-brain: analysis’

- What made A fail ?
  - Once A and B came back up as a new replication chain, they had outdated data
  - If B would have come back before A, it could have been re-slaved to X
  - But because A came back before re-slaving, it injected heartbeat and p-GTID to B
  - Then B could have been re-cloned without problems

```
+----+
| A  |
+----+
  |
+----+      +----+     Master RWs
| B  |      | X  | <-- happen here
+----+      +----+
              |
            +----+     Reads
            | Y  | <-- happen here
            +----+
```

Split-brain: analysis’

- What made A fail ?
  - Once A and B came back up as a new replication chain, they had outdated data
  - If B would have come back before A, it could have been re-slaved to X
  - But because A came back before re-slaving, it injected heartbeat and p-GTID to B
  - Then B could have been re-cloned without problems
  - But A was re-cloned instead (**human error #1**)

```
+----+
| A  |
+----+

+\-/+     +----+     Master RWs
| B  |    | X  | <-- happen here
+/-\+     +----+
            |
          +----+     Reads
          | Y  | <-- happen here
          +----+
```

Split-brain: analysis'

- What made A fail ?
  - Once A and B came back up as a new replication chain, they had outdated data
  - If B would have come back before A, it could have been re-slaved to X
  - But because A came back before re-slaving, it injected heartbeat and p-GTID to B
  - Then B could have been re-cloned without problems
  - But A was re-cloned instead (**human error #1**)
- Why did Orchestrator not fail-over right away ?
  - B was promoted hours after A was brought down…
  - Because A was downtimed only for 4 hours (**human error #2**)

```
+\-/+
| A |
+/-\+

+---+     +---+     Master RWs
| B |     | X | <-- happen here
+---+     +---+
            |
          +---+     Reads
          | Y | <-- happen here
          +---+
```

Orchestrator anti-flapping

- Orchestrator has a failover throttling/acknowledgment mechanism[1]:
  - Automated recovery will happen
    - for an instance in a cluster that has not recently been recovered
    - unless such recent recoveries were acknowledged
- In our case:
  - the recovery might have been acknowledged too early (**human error #0 ?**)
  - with a too short “recently” timeout
  - and maybe Orchestrator should not have failed over the second time

[1]: https://github.com/github/orchestrator/blob/master/docs/topology-recovery.md #blocking-acknowledgments-anti-flapping

Split-brain: summary

- So in summary, this disaster was caused by:
  1. A fancy failure: two servers failing at the same time
  2. A debatable premature acknowledgment in Orchestrator
     and probably too short a timeout for recent failover
  3. Edge-case recovery: both servers forming a new replication topology
  4. Restarting with the event-scheduler enabled (A injecting heartbeat and p-GTID)
  5. Re-cloning wrong (A instead of B; should have created C and D and thrown away A and B; too short downtime for the re-cloning)
  6. Orchestrator failing over something that it should not have
     (including taking an questionnable action when a downtime expired)
  7. DNS repointing script not defensive enough

Fancy failure: more details

- Why did A and B fail at the same time ?
  - Deployment error: the two servers in the same rack/failure domain ?
  - And/or very unlucky ?

```
+\-/+
| A |
+/-\+


+\-/+     +----+
| B |     | X  |
+/-\+     +----+
            |
          +----+
          | Y  |
          +----+
```

- Very unlucky because…

  10 to 20 servers failed that day in the same data center

  Because human operations and “sensitive” hardware

How to fix such situation ?

- Fixing “split-brain” data on B and X is hard
- Some solutions are:
  - Kill B or X (and lose data)
  - Replay writes from B on X or vice-versa (manually or with replication)
  - But AUTO_INCREMENTs are in the way:
    - up to i used on A before 1st failover
    - i-n to j_1 used on X after recovery
    - i to j_2 used on B after 2nd failover

```
                 +\-/+
                 | A |
                 +/-\+

Master RWs       +---+     +---+
happen here -->  | B |     | X |
                 +---+     +---+
                             |
                           +---+     Reads
                           | Y | <-- happen here
                           +---+
```

Takeaway

- Twisted situations happen
- Automation (including failover) is not simple:
  → code automation scripts defensively
- Be mindful for premature acknowledgment, downtime more than less, shutdown slaves first → understand complex interactions of tools in details
- Try something else than AUTO-INCREMENTs for Primary Key (monotonically increasing UUID[1] [2] ?)

[1]: https://www.percona.com/blog/2014/12/19/store-uuid-optimized-way/
[2]: http://mysql.rjweb.org/doc.php/uuid

Re Failover War Story

Master failover does not always go as planned

- because it is complicated

It is not a matter of “if” things go wrong

- but “when” things will go wrong

Please share your war stories

- so we can learn from each-others’ experience
- GitHub has a MySQL public Post-Mortem (great of them to share this): https://blog.github.com/2018-10-30-oct21-post-incident-analysis/

MessageBird is hiring

messagebird.com/en/careers/

Thanks !

by Jean-François Gagné
presented at SRECon Dublin (October 2019)
Senior Infrastructure Engineer / System and MySQL Expert
jeanfrancois AT messagebird DOT com / @jfg956 #SREcon