Booking.com

How to setup Orchestrator to manage thousands of MySQL servers

Simon J Mudd | 3rd October 2017

Session Summary

- What is orchestrator and why use it?
- What happens as you monitor more servers?
- Features added to make it scale and improve usability
- Using orchestrator at smaller scale
- Way forward

Booking.com

- One of the largest travel e-commerce sites in the world
  - part of the Priceline Group (NASDAQ: PCLN)
  - We offer accommodation in 228 countries
  - Our website and customer service in 40 languages
  - More than 15,000 employees in 204 offices in 70 countries
- We use thousands of MySQL servers:
  - we use orchestrator to manage the topology and handle master and intermediate master failures

What is Orchestrator and why use it?

Orchestrator

orchestrator

<> with ♥ by GitHub

Booking.com

Orchestrator

- Written by Shlomi Noach
  - he started on this at outbrain and is now working at github.com
  - He introduced booking.com to orchestrator about 3 years ago when we were looking for something to handle failovers automatically

Orchestrator

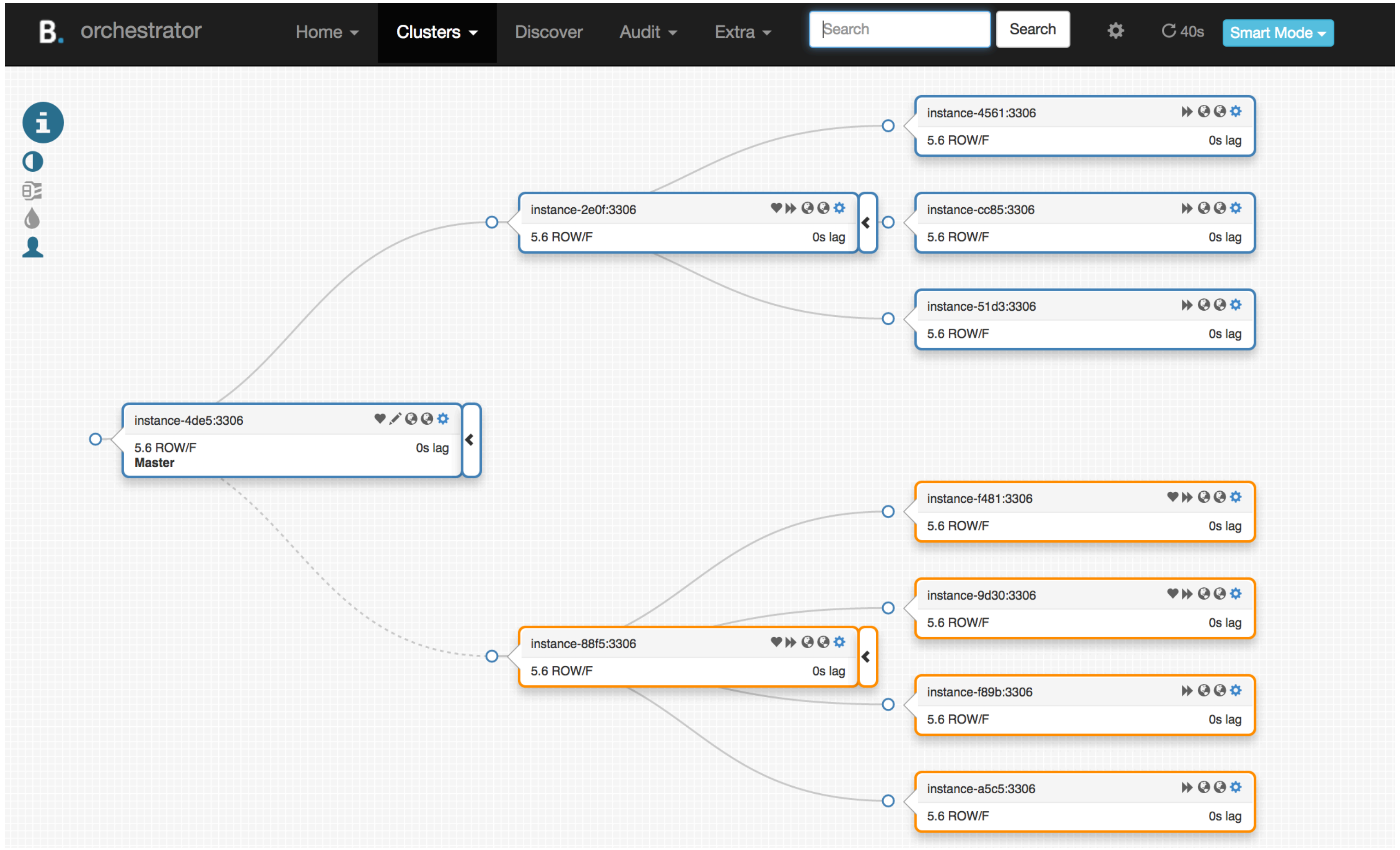
- Periodically monitors MySQL servers and checks their health
- Handles **master failover**, but also does much more…
- GUI to manage and visualise topology – very handy
- CLI to do the same tasks – used for scripting
- API calls to run at a distance
- Needs a DB backend to store state.
  - Normally MySQL but can be SQLite
- Written in go

Orchestrator

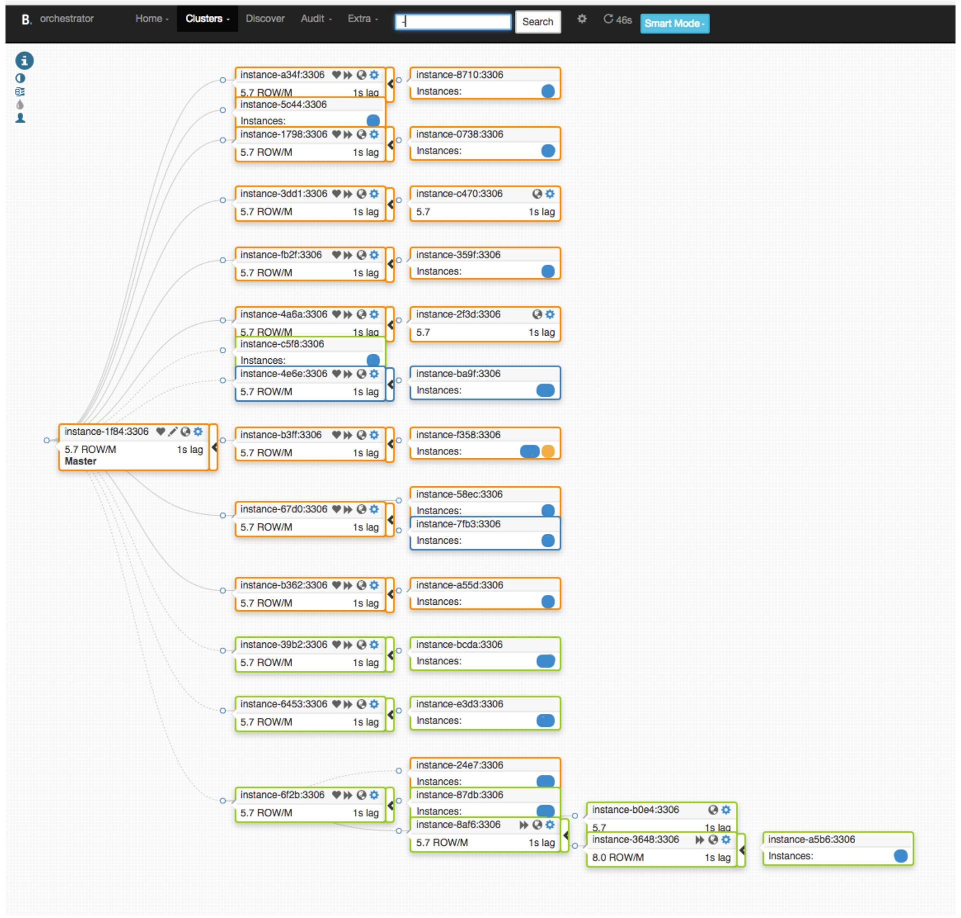
What failures does it handle?

- Master failures
  - Optional hooks to external systems which need to be aware of these failures
- Intermediate master failures
- Does **not** care about leaf slaves or applications
- Works with Oracle or MariaDB GTID
- Works without GTID: Can add *Pseudo-GTID* (events injected on the master are used to find a match) so no *need* to migrate
- Handles multi-level topologies

Orchestrator GUI

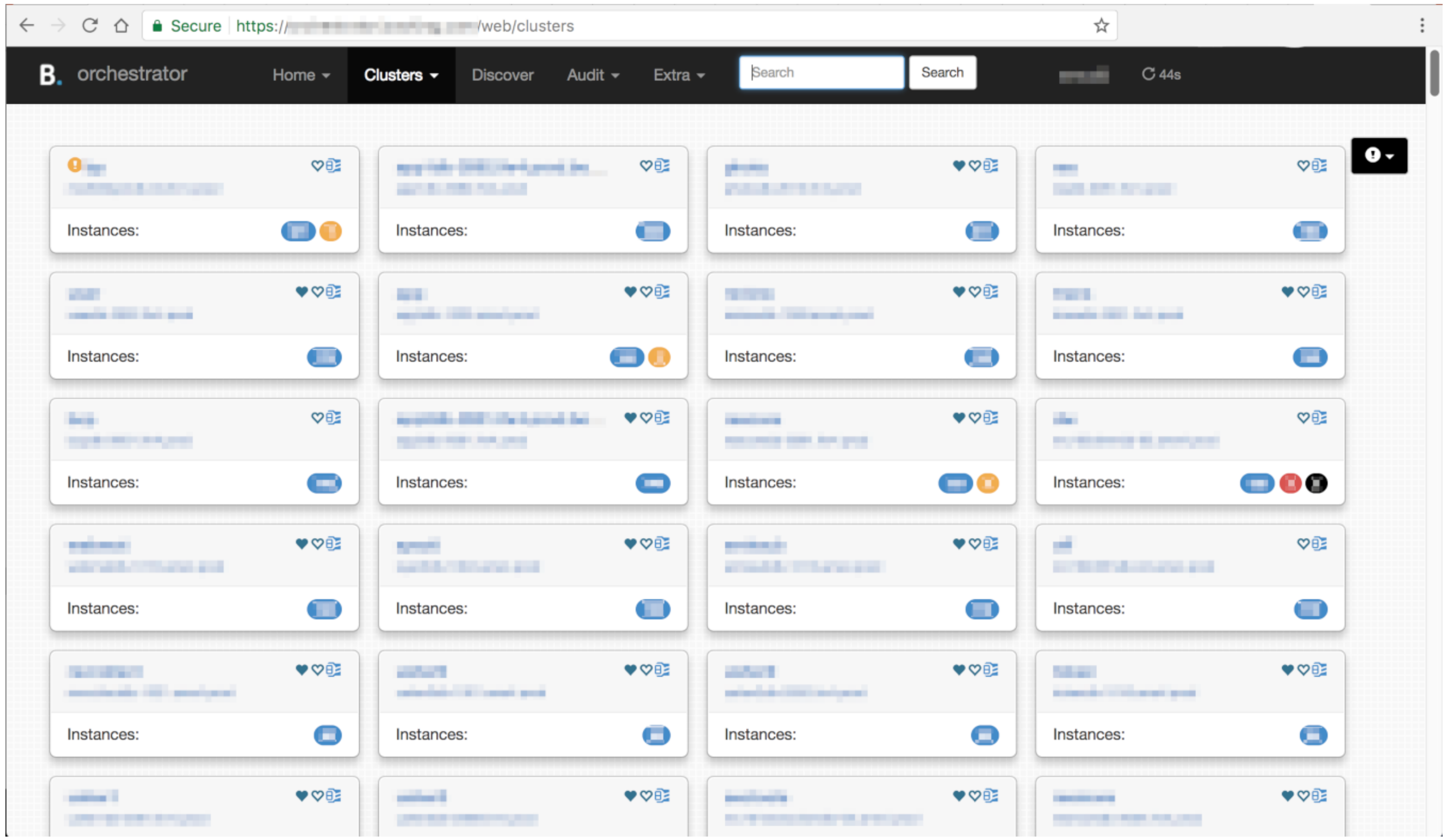


Orchestrator GUI



Booking.com

Orchestrator GUI



Booking.com

Orchestrator

Topology Management

- Drag and drop using the GUI
  - Move one slave about
  - Move all slaves
- Scriptable relocation from the command line or using API calls

What happens as you monitor more servers?

What happens as you monitor more servers?

- Integration needed with internal infrastructure
  - Deployment: tell orchestrator to discover and forget servers*
  - Determine candidate masters
  - Handle special cases:
    - test MySQL versions, special setups (black- or white-list servers or clusters)
- Make orchestrator HA
- Monitor orchestrator behaviour and performance
- Provide wider access to different types of user

* It can automatically detect new servers in an existing cluster but not new detect new clusters without help

Integration with Internal Infrastructure

- Populate the metadata db on the master to:
  - Map host or instance names to more familiar cluster names
  - How to determine replication delay
  - Configuration of acceptable levels of replication delay
- Add and removal of servers/instances as they are deployed or removed from service
- Setup of Pseudo-GTID (if not using GTID)

Integration with Internal Infrastructure

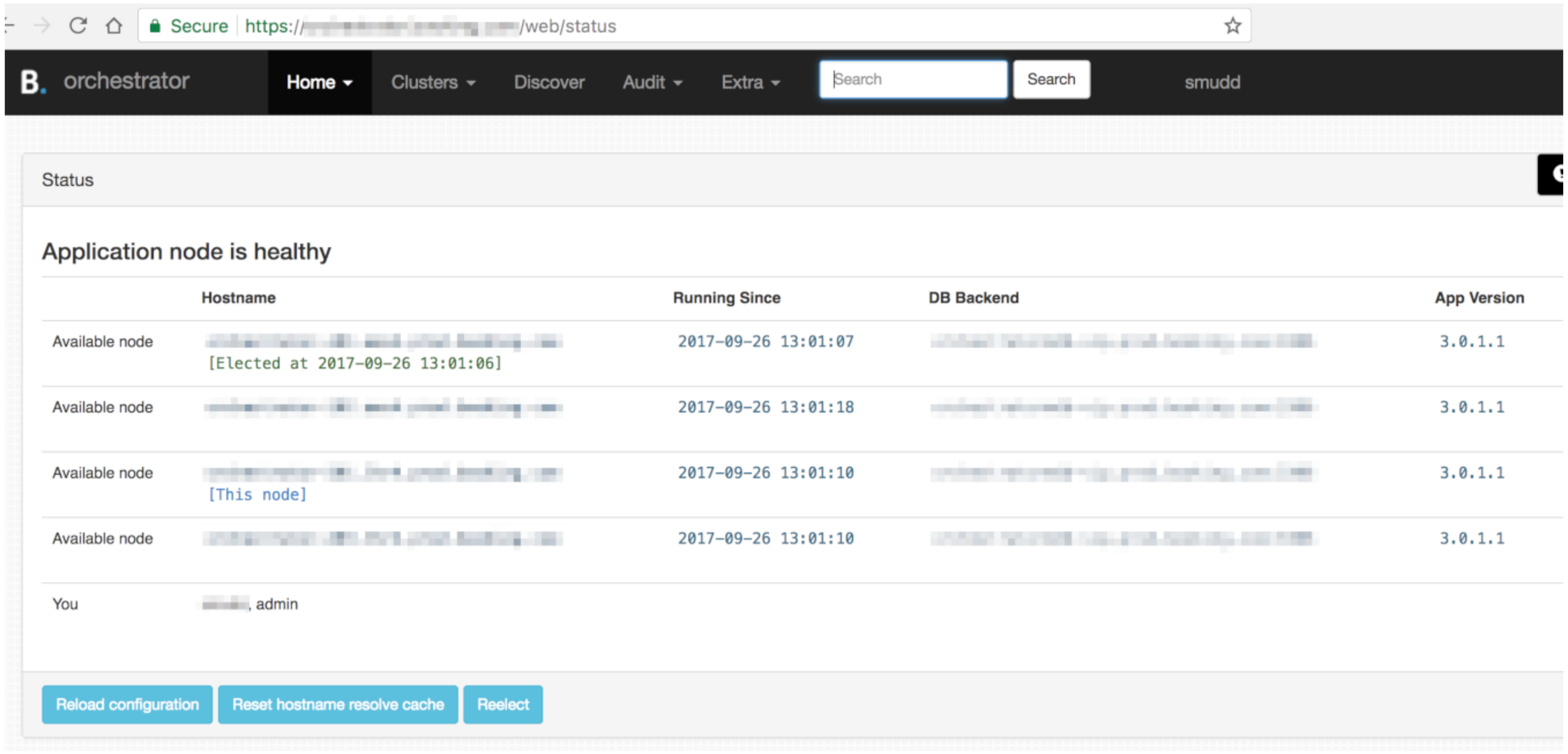
- Add failover hooks for monitoring, notification and to take site-specific actions (tell other systems about the new master)
- Selection of candidate masters
- Blacklisting servers which are not suitable: backup servers, test servers, servers in the wrong network areas …

Better Visibility

- Improve orchestrator deployment visibility
  - For each running app: show host, version, uptime
  - Show the active node and how long it's been active
- Auditing of MySQL failures and recovery via the GUI is good and improving
  - no need to search the logs

Booking.com

Better Visibility



Status

Application node is healthy

	Hostname	Running Since	DB Backend	App Version
Available node	[redacted] [Elected at 2017-09-26 13:01:06]	2017-09-26 13:01:07	[redacted]	3.0.1.1
Available node	[redacted]	2017-09-26 13:01:18	[redacted]	3.0.1.1
Available node	[redacted] [This node]	2017-09-26 13:01:10	[redacted]	3.0.1.1
Available node	[redacted]	2017-09-26 13:01:10	[redacted]	3.0.1.1
You	[redacted], admin			

Features added to scale and improve usability

Performance

We found bottlenecks especially on startup

- Try to discover several thousand mysql servers at once and update the backend at the same time → max_connections exceeded
- Multiple go routines trying to poll the same stuck server

Solution:

- FIFO Discovery queue which avoids duplicates and limits maximum discovery concurrency

Performance

How to figure out what’s going on?

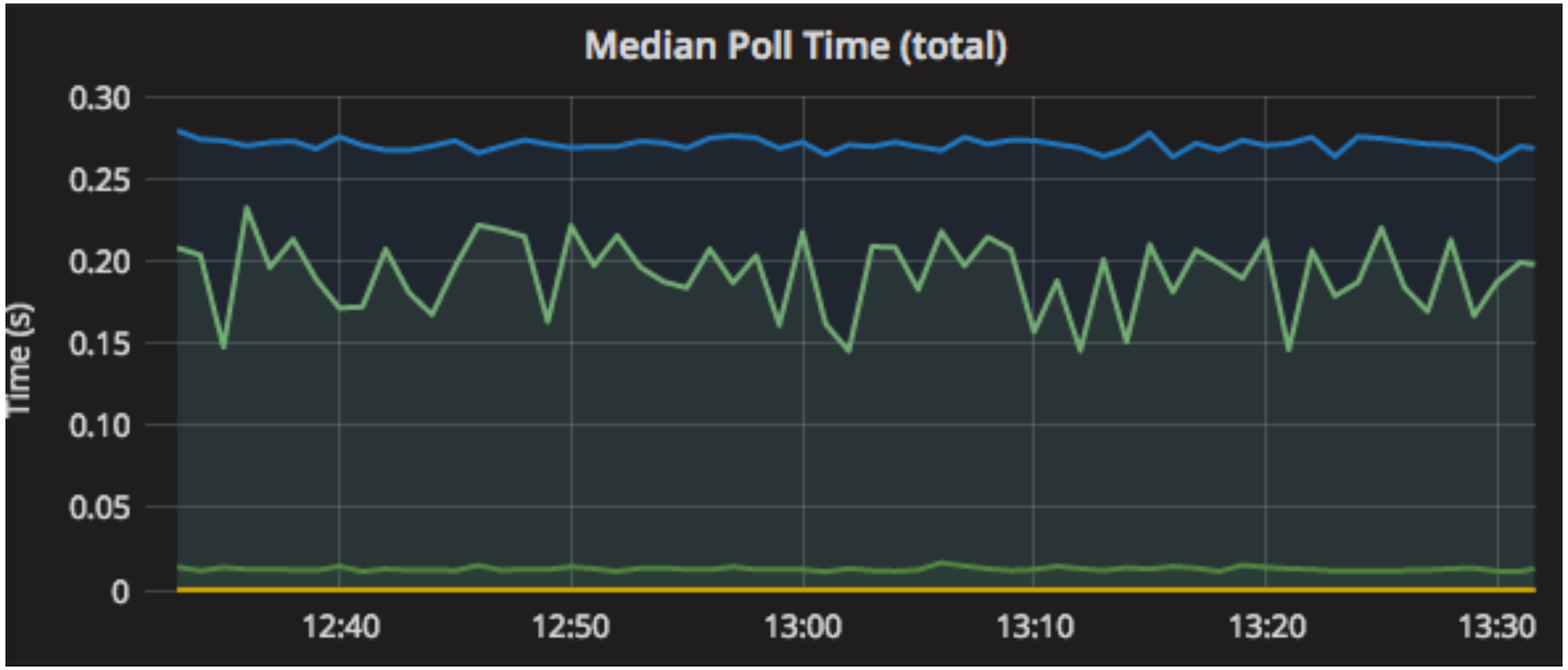
- Understanding logging is hard at this scale – too much noise
- No discovery metrics to see problems at server or aggregate level

Solution:

- Collect discovery metrics and keep for N seconds
- Log discovery times in debug mode
- Provide interface to retrieve raw or aggregate values to use in monitoring systems

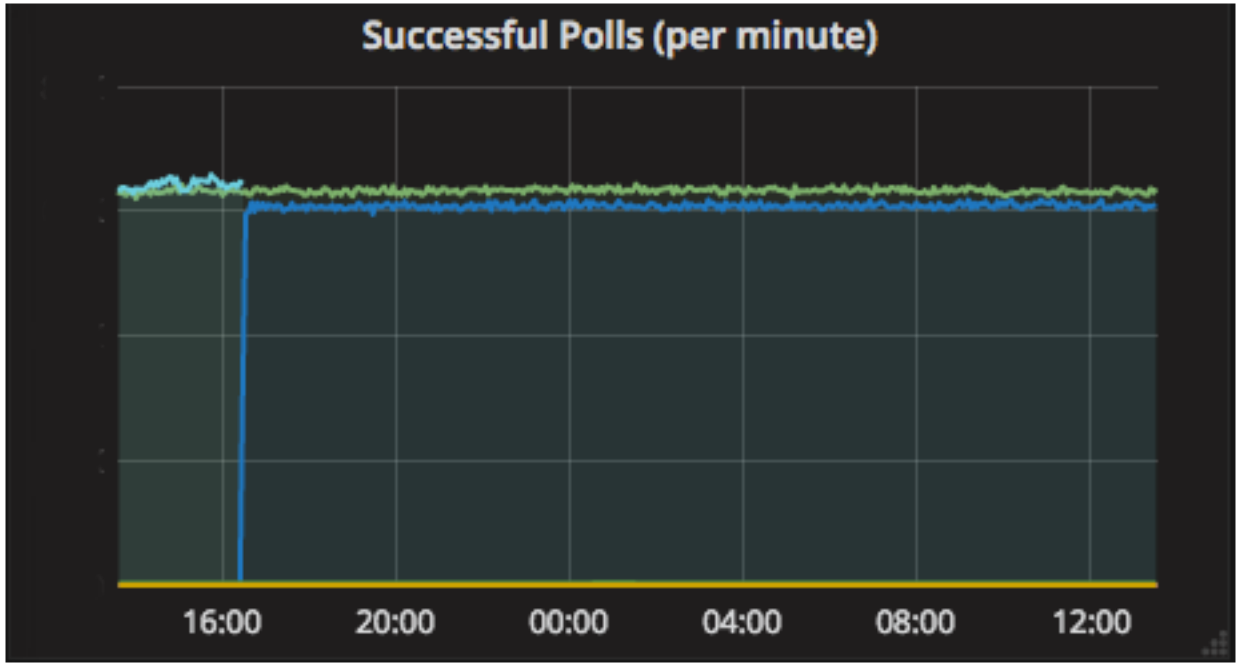
Booking.com

Performance



Discovery (Poll) times

Performance



Discovery (Poll) counters

Booking.com

Performance

- A client upgrade might upgrade the database which other older apps were still using

Solution:

- Make auto-upgrade of the database optional so the DBA controls this

Booking.com

Performance

- Cross zone (dc) access changes performance profile significantly and caused problems
  - orchestrator apps are supposed to be easy to replace and location should not matter
  - latency can be a real enemy

Solution:

- Batch updates of some data into smaller number of larger inserts
- Collect metrics on these timings
- Catch discoveries which take too long (internal code bottlenecks)
- Visibility of the metrics made it easier to locate causes

Booking.com

Performance

- Special connections settings
  - "MySQLOrchestratorMaxPoolConnections": control go pool size
  - "MySQLConnectTimeoutSeconds": 1
    - don’t waste time waiting to connect to a dead server

Booking.com

Performance

golang specific *-isms*

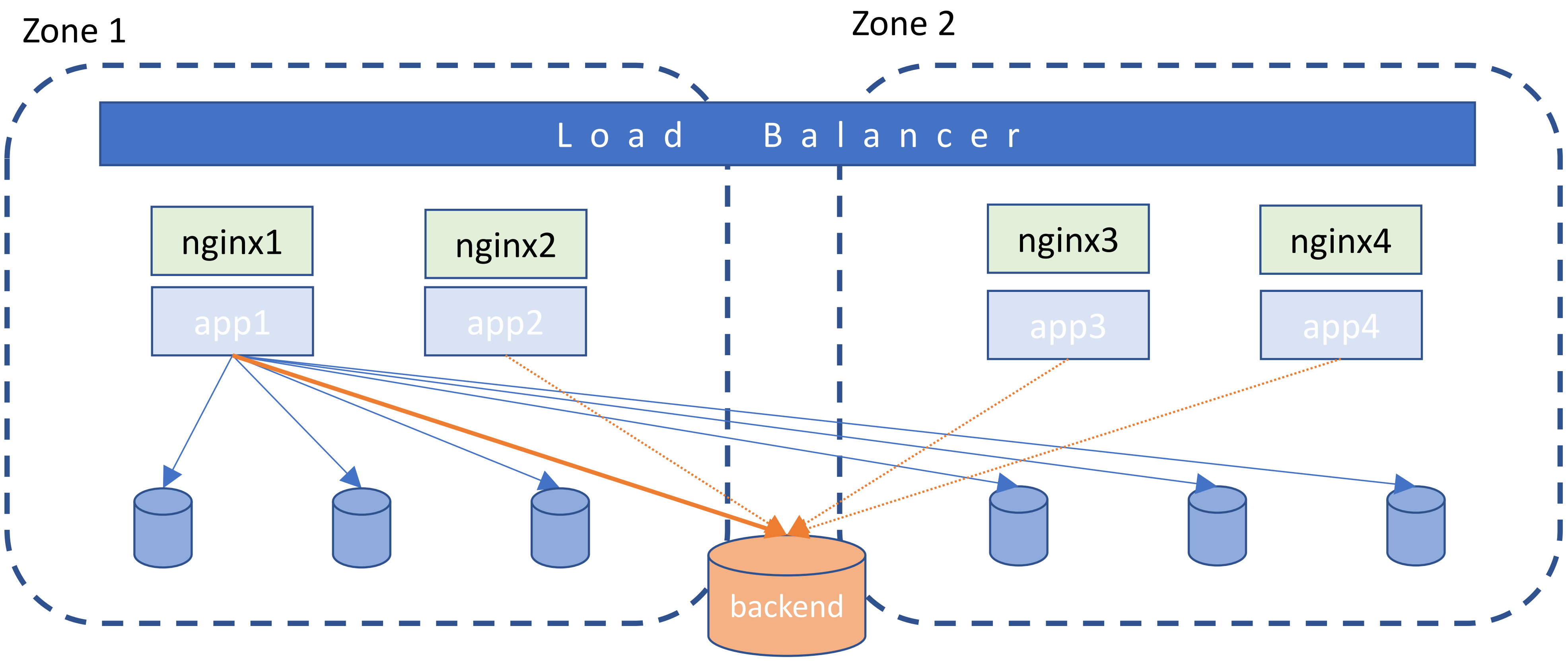
- Orchestrator by default uses database/sql and by default sends a query with parameters using MySQL's Prepare/Execute syntax
  - This generates 2 rtt's and on slower connections can affect the elapsed time to complete a query
  - Options to disable this by interpolating parameter values prior to sending SQL
- Go (orchestrator code) is quite happy to *try* to poll 10,000 servers at once
  - Sometimes that is not sensible
  - Throttling to avoid thundering herd is necessary

Booking.com

Orchestrator HA

- More then one orchestrator server per zone/dc
  - Some upgrades really easy – just restart with new binaries
- Common end point via **load balancer**
  - Simpler for users
  - works for api calls and may simplify firewall rules

Orchestrator HA



Booking.com

Orchestrator HA

Might I have more than one orchestrator cluster?

- Yes for active development
  - as a side-effect gives us extra redundancy
  - Development load is too small to catch many issues
  - Recoveries disabled **globally** on this cluster but monitoring works the same
- Compliance regulations may require segregation of different networks

Orchestrator HA

Solution

- Move from using orchestrator binary to use cluster API interface
  - Recently migrated to use new orchestrator-client command which solves the same problem and was needed for orchestrator/raft access
- Simplifies configuration
- Allows easy access to more then one orchestrator cluster
- Orchestrator upgrades with db backend changes are easier

Orchestrator API

Enhancements to API calls

- Bulk retrieval of instance information and promotion rules
- Asynchronous discovery call (e.g. bootstrap new cluster)
- More monitoring information available
  - Discovery timing metrics
  - Discovery queue metrics
  - Backend write metrics

Booking.com

Special Cases

- Testing MySQL 8.0 or MariaDB 10.3?
  - "Let's not promote to this box"
  - Same applies while testing new minor versions of course
- Some topologies have slaves with aggregate data
  - **Do not** treat them as a normal box – **should not** be candidate masters
- Orchestrator can not handle GR or multi-source replication yet
  - Best to **avoid** these boxes (for automatic failover) until we have solutions
  - Patches welcome to solve such missing functionality

Special Cases

Handling TLS connections

- Orchestrator could handle using TLS or not using it but …
- Some servers need to be accessed by TLS, others don't (ODBC access or more security *sensitive* systems)
- Orchestrator could not handle this
- Code added to recognise error and automatically switch to TLS:
  - `Error 3159: Connections using insecure transport are prohibited while --require_secure_transport=ON`
- Global **OFF** button – gives you peace of mind

Provide Wider User Access

- Orchestrator fan club
  - Different groups of users like orchestrator
  - DBAs, Developers, Sysadmins, Auditors, Managers
- Use nginx (or similar)
  - Provides authentication
  - Provides TLS
  - The combination can be used with unix groups to allow *user* or *admin* access to orchestrator
- Combined with a load balancer provides easy access for users and also for applications (using api calls)

Monitoring

Some things to monitor

- Orchestrator process (and nginx)
- Orchestrator cluster endpoint
- Successful or failed Discoveries per minute
- Discovery queue sizes
- Discovery timings
  - aggregate data gives mean, median and percentiles
- Discoveries exceeding InstancePollSeconds
- When changing active orchestrator node these values <u>may</u> change

Booking.com contributions

Commits to public orchestrator repo

- Simon: 170
- Dmitry: 40
- Mauro: 15
- Daniël: 8
- Shlomi: many (while working at booking)

Using orchestrator at smaller scale

Booking.com

Using orchestrator at smaller scale

Not mentioned here but

- Consider use of Sqlite – good starting point – single binary
- Consider use of Sqlite/raft
  - provides HA
  - **all nodes** monitor all MySQL servers
- Only difference is the db backend
- Not sure where scaling limits

Booking.com

Configuration settings

Settings to be considered, broken down by function

Configuration settings

- MySQL backend
    - "MySQLOrchestratorCredentialsConfigFile": "/path/.my-orchestratordb.cnf"
    - "MySQLOrchestratorDatabase": "orchestrator”
    - "MySQLOrchestratorHost": "orchestratordb.example.com"
    - "MySQLOrchestratorPort": 3306
    - "MySQLOrchestratorMaxPoolConnections": 100
    - "MySQLConnectTimeoutSeconds": 1
- Sqlite backend
    - "BackendDB": "sqlite”
    - "SQLite3DataFile": "/var/lib/orchestrator/orchestrator.db"

Configuration settings

- Psuedo-GTID Settings (if using pseudo-gtid)
  - PseudoGTIDPattern
  - PseudoGTIDMonotonicHint
  - DetectPseudoGTIDQuery

Booking.com

Configuration settings

- Cluster and host settings
  - Query metadata db (populated externally) to detect clusters
  - DetectClusterAliasQuery
  - DetectClusterDomainQuery

Configuration settings

- Recovery settings
  - Regexp filters – very site dependent
  - RecoverMasterClusterFilters – white-list masters by cluster name
  - RecoverIntermediateMasterClusterFilters
  - PromotionIgnoreHostnameFilters – ignore servers from being promoted*
  - RecoveryIgnoreHostnameFilters – ignore special servers from recovery

* Does not scale well

Booking.com

Configuration settings

- Failover settings
  - OnFailureDetectionProcesses – what to do when a failure is detected
  - PreFailoverProcesses – what to do prior to starting recovery
  - PostFailoverProcesses – what to do after completing recovery
  - PostUnsuccessfulFailoverProcesses – what to do if recovery fails
  - PostMasterFailoverProcesses – what to do after IM recovery
  - PostIntermediateMasterFailoverProcesses – what to do after Master recovery

Configuration settings

- Authentication settings (e.g. if using nginx with LDAP)
  - "AuthenticationMethod": "proxy",
  - "HTTPAuthUser": ”user1",
  - "HTTPAuthPassword": ”pass1",
  - "AuthUserHeader": ”SomeHeader",
  - “PowerAuthUsers": ["api-user1", "api-user2", ”realuser1"]
  - PowerAuthGroups": [”special_sysadmins”, “dbas”],

* Does not scale well

Configuration settings

- Environment settings (e.g. shorten/simplify hostnames)
  - “DataCenterPattern”
  - “PhysicalEnvironmentPattern”:
  - “RemoveTextFromHostnameDisplay”: “:.example.com:3306”

Way Forward

Booking.com

Way Forward

- Improvements needed to tackle problems at both ends of the scale
  - Smaller installations – for getting on board
  - Larger installations – to allow for further scaling

Way Forward

- Simplify configuration and entry to orchestrator
  - Shlomi is doing a very good job with sqlite and raft setups
  - Configuration could be simpler and more automatic for most people
  - Need to standardise orchestrator setups more?
- Extend functionality to cover more of the MySQL eco-system
  - AWS and other cloud systems
  - Group Replication or Galera
  - Multi-source

Way Forward

- Distribution of discoveries amongst all orchestrator nodes
  - Orchestrator/raft: all nodes monitor all MySQL servers
    - Raft usage recommends having several nodes
  - Orchestrator/MySQL: one node monitors all MySQL servers
  - **Better:** distribute monitoring amongst available nodes
    - Avoids unnecessary load on monitored servers
    - reduces work on busy orchestrator apps
    - Useful for small and large installations
  - efficient balancing is harder

Way Forward

- Reduce recovery time
  - Speeding up *detection to recovery time* would be good as <u>reduces downtime</u>
  - Should be possible to react to failure event (knowing state of other servers) *immediately*
    - state currently stored in backend db
    - analysis and detection phase happens independently of server polling
  - With reduced default poll time of 5 seconds recovery is likely to be *triggered* within 10 seconds
  - not critical for most people?

Way Forward

- Further work needed to scale more
  - bottlenecks still exist
  - Larger installations keep growing
- Improve monitoring
  - External API calls
  - Add internal metrics

Conclusion

Booking.com

Does it work?

I checked for failures over a recent period

- 6 master failures
- About 40 intermediate master failures
- No-one called up
- No harm was done

Questions?

Thanks

Simon J Mudd
simon.mudd@booking.com