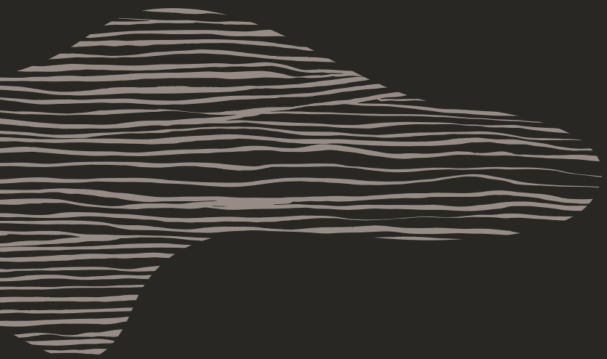
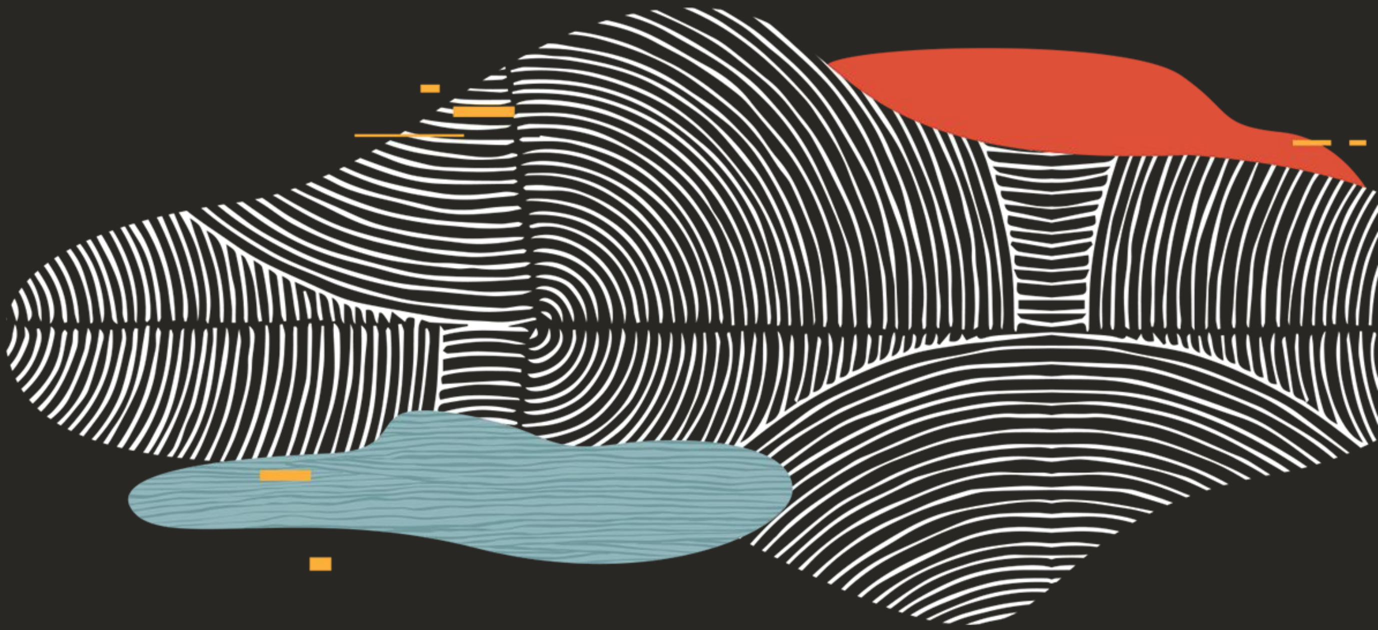



MySQL 8.0 Operational Changes

Dave Stokes

Community Manager, North America

MySQL Community Team

Safe Harbor

MySQL 8.0 is available in cPanel and WHM v88!!

V88
& MySQL 8.0!!!

Our Goal

MySQL 8.0 was a big advancement over the previous version, MySQL 5.7 and this presentation is a general overview of some of the new features.

There are many new facets not covered by this presentation and hopefully you will be able to explore them later.

MySQL 5.6

If you are using MySQL 5.6 please note that it reaches End of Life status for support and updates in February 2021.

Please upgrade if you are using this version of MySQL!

What is MySQL?

25 Years old!!

- Relational Database Management System
- GPLv2 licensed Community Edition
- Licensed Enterprise Edition

  Monitoring software, improved backup, at rest encryption, data masking, db firewall, key chain management, and support

- Also a NoSQL JSON Document Store Database
- Also language connectors, Router, Shell, InnoDB Cluster
- And more!

What Happened to MySQL 6.0 and 7.0????

There was an early MySQL 6.0 that withered on the vine and software was included in the 5.0 series

NDB Cluster has used the 7 series for many years

So engineering thought we earned the 8.0 series which became generally available in April 2018

A few months ago I did a presentation on new facets that was mainly developer oriented and this talk is more DevOps-centric.

Big Changes in 8.0

Resource groups -- Dedicate cores to classes of queries, denote in comments

Faster backup -- MySQL Shell utilities **dumpSchema** & **dumpInstance**

Compression -- Replication logs compressed and kept that way

New Temp Table, UTF8MB4 optimized, & more!

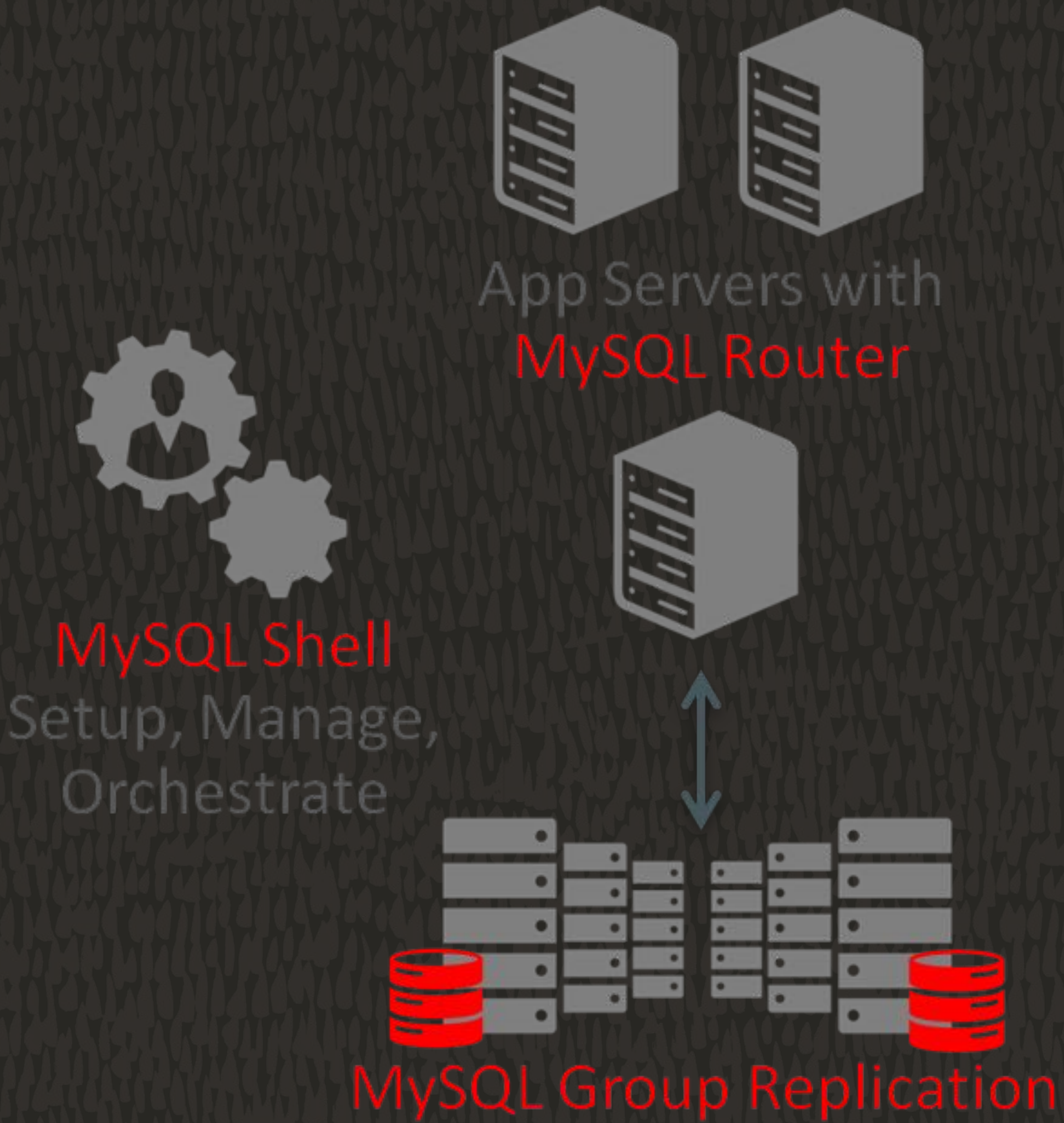
CI/CD Model

New releases of MySQL every few months:

- 8.0.21 July 13 2019
- 8.0.20 April 27
- 8.0.19 January 13
- 8.0.18 October 15 2018
- 8.0.17 July 22

Replication - Where Most DevOps folks seem to live

1. InnoDB Cluster
2. Replica Sets
3. Router



InnoDB Cluster Components

- **Router** is a lightweight Level 4 router designed to be hosted on your application platform
- **Group Replication** creates elastic, highly available, fault tolerant replication topologies. Either single-primary multi-primary modes.
-
- **MySQL Shell** used for administration

Sandbox Test

Please start with a sandbox cluster to see just how easy it is to set up InnoDB Cluster

1. Start the MySQL Shell

```
$ mysqlsh root@localhost
```

2. Deploy sandbox first node

```
JS > dba.deploySandboxInstance(3310)
A new MySQL sandbox instance will be created on this host in
/home/dstokes/mysql-sandboxes/3310

Warning: Sandbox instances are only suitable for deploying and
running on your local machine for testing purposes and are not
accessible from external networks.

Please enter a MySQL root password for the new instance: ******

Deploying new MySQL instance...

Instance localhost:3310 successfully deployed and started.
Use shell.connect('root@localhost:3310') to connect to the instance.
```

3. Deploy sandbox second node

```
JS > dba.deploySandboxInstance(3320)
A new MySQL sandbox instance will be created on this host in
/home/dstokes/mysql-sandboxes/3320

Warning: Sandbox instances are only suitable for deploying and
running on your local machine for testing purposes and are not
accessible from external networks.

Please enter a MySQL root password for the new instance: ******

Deploying new MySQL instance...

Instance localhost:3320 successfully deployed and started.
Use shell.connect('root@localhost:3320') to connect to the instance.
```

4. Deploy sandbox third node

```
JS > dba.deploySandboxInstance(3330)
A new MySQL sandbox instance will be created on this host in
/home/dstokes/mysql-sandboxes/3330

Warning: Sandbox instances are only suitable for deploying and
running on your local machine for testing purposes and are not
accessible from external networks.

Please enter a MySQL root password for the new instance: ******

Deploying new MySQL instance...

Instance localhost:3330 successfully deployed and started.
Use shell.connect('root@localhost:3330') to connect to the instance.
```

5. Connect to first node

```
JS > \connect root@localhost:3310
Creating a session to 'root@localhost:3310'
Please provide the password for 'root@localhost:3310': ******
Save password for 'root@localhost:3310'? [Y]es/[N]o/Ne[v]er (default No): y
Fetching schema names for autocompletion... Press ^C to stop.
Closing old connection...
Your MySQL connection id is 12
Server version: 8.0.20 MySQL Community Server - GPL
No default schema selected; type \use <schema> to set one.
```

6. Create cluster

```
JS > var cluster = dba.createCluster('cpanel')
A new InnoDB cluster will be created on instance 'localhost:3310'.

Validating instance configuration at localhost:3310...
NOTE: Instance detected as a sandbox.
Please note that sandbox instances are only suitable for deploying test clusters for use within the same host.

This instance reports its own address as 127.0.0.1:3310

Instance configuration is suitable.
NOTE: Group Replication will communicate with other members using '127.0.0.1:33101'. Use the localAddress option to override.

Creating InnoDB cluster 'cpanel' on '127.0.0.1:3310'...

Adding Seed Instance...
Cluster successfully created. Use Cluster.addInstance() to add MySQL instances.
At least 3 instances are needed for the cluster to be able to withstand up to one server failure.
```

7. Add second node to cluster

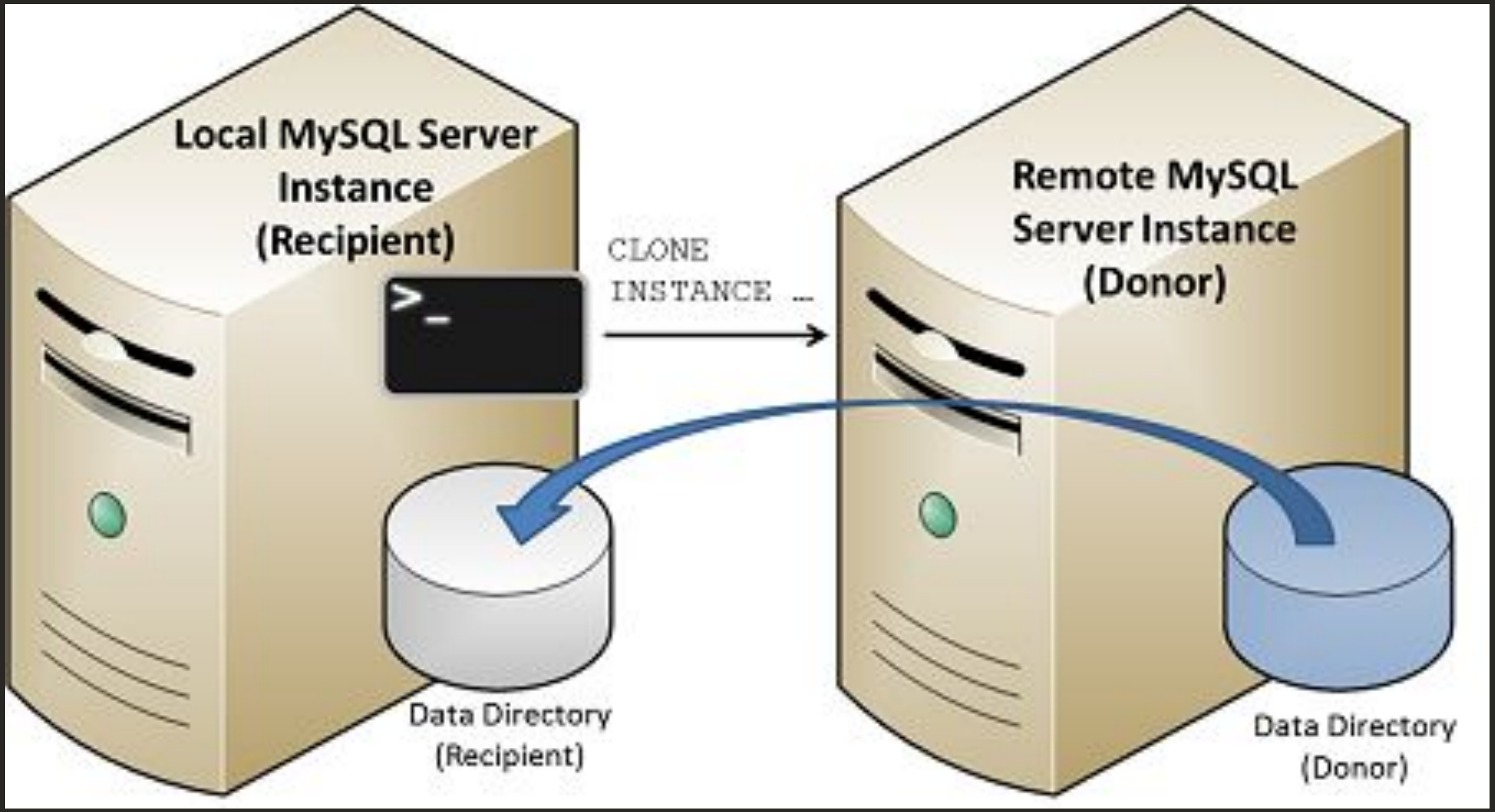
```
JS > cluster.addInstance('root@localhost:3320')

Please provide the password for 'root@localhost:3320': ******

Save password for 'root@localhost:3320'? [Y]es/[N]o/Ne[v]er (default No): yes


NOTE: The target instance '127.0.0.1:3320' has not been pre-provisioned (GTID set is empty).
The Shell is unable to decide whether incremental state recovery can correctly provision it.

The safest and most convenient way to provision a new instance is through automatic clone
provisioning, which will completely overwrite the state of '127.0.0.1:3320' with a physical
snapshot from an existing cluster member. To use this method by default, set the
'recoveryMethod' option to 'clone'.


The incremental state recovery may be safely used if you are sure all updates ever executed in
the cluster were done with GTIDs enabled, there are no purged transactions and the new
instance contains the same GTID set as the cluster or a subset of it. To use this method by
default, set the 'recoveryMethod' option to 'incremental'.
```

7. Add second node to cluster continued

```
Please select a recovery method [C]lone/[I]ncremental
recovery/[A]bort (default Clone): C

NOTE: Group Replication will communicate with other members using '127.0.0.1:33201'. Use the localAddress
option to override.


Validating instance configuration at localhost:3320...

NOTE: Instance detected as a sandbox.

Please note that sandbox instances are only suitable for deploying test clusters for use within the same host.


This instance reports its own address as 127.0.0.1:3320


Instance configuration is suitable.

A new instance will be added to the InnoDB cluster. Depending on the amount of

data on the cluster this might take from a few seconds to several hours.
```

7. Add second node to cluster continued

```
Adding instance to the cluster...

Monitoring recovery process of the new cluster member. Press ^C to stop monitoring and let it continue in background.
Clone based state recovery is now in progress.

NOTE: A server restart is expected to happen as part of the clone process. If the
server does not support the RESTART command or does not come back after a
while, you may need to manually start it back.
* Waiting for clone to finish...
NOTE: 127.0.0.1:3320 is being cloned from 127.0.0.1:3310
** Stage DROP DATA: Completed
** Clone Transfer
    FILE COPY  ############################################################  100%  Competed
    PAGE COPY  ############################################################  100%  Completed
   REDO COPY  ############################################################  100%  Completed
NOTE: 127.0.0.1:3320 is shutting down...
A new instance will be added to the InnoDB cluster. Depending on the amount of data on the cluster this might take from a few seconds to several hours.
```


Local MySQL Server Instance (Recipient)
>_
CLONE INSTANCE ...
Remote MySQL Server Instance (Donor)
Data Directory (Recipient)
Data Directory (Donor)

The clone plugin permits cloning data locally or from a remote MySQL server instance.

Cloned data is a physical snapshot of data stored in InnoDB that includes schemas, tables, tablespaces, and data dictionary metadata.

7. Add second node to cluster continued

```
* Waiting for server restart... ready
* 127.0.0.1:3320 has restarted, waiting for clone to finish...
** Stage RESTART: Completed
* Clone process has finished: 59.62 MB transferred in about 1 second (~59.62 MB/s)

Incremental state recovery is now in progress.

* Waiting for distributed recovery to finish...
NOTE: '127.0.0.1:3320' is being recovered from '127.0.0.1:3310'
* Distributed recovery has finished
```

8. Add third node to cluster continued

```
JS > cluster.addInstance('root@localhost:3330')
JS > cluster.status()
{
    "clusterName": "cpanel",
    "defaultReplicaSet": {
        "name": "default",
        "primary": "127.0.0.1:3310",
        "ssl": "REQUIRED",
        "status": "OK",
        "statusText": "Cluster is ONLINE and can tolerate
up to ONE failure.",
        "topology": {
            "127.0.0.1:3310": {
                "address": "127.0.0.1:3310",
                "mode": "R/W",
                "readReplicas": {},
                "replicationLag": null,
                "role": "HA",
                "status": "ONLINE",
                "version": "8.0.20"
            },
            "127.0.0.1:3320": {
                "address": "127.0.0.1:3320",
                "mode": "R/O",
                "readReplicas": {},
                "replicationLag": null,
                "role": "HA",
                "status": "ONLINE",
                "version": "8.0.20"
            },
            "127.0.0.1:3330": {
                "address": "127.0.0.1:3330",
                "mode": "R/O",
                "readReplicas": {},
                "replicationLag": null,
                "role": "HA",
                "status": "ONLINE",
                "version": "8.0.20"
            }
        },
        "topologyMode": "Single-Primary"
    },
    "groupInformationSourceMember":
"127.0.0.1:3310"
```

9. Start Router

```
$ mysqlrouter --bootstrap root@localhost:3310 --directory
/tmp/myrouter&
Please enter MySQL password for root:
# Bootstrapping MySQL Router instance at '/tmp/myrouter'...

- Creating account(s) (only those that are needed, if any)
- Verifying account (using it to run SQL queries that would be run by Router)
- Storing account in keyring
```

10. Start Router continued

```
- Adjusting permissions of generated files
- Creating configuration /tmp/myrouter/mysqlrouter.conf


# MySQL Router configured for the InnoDB Cluster 'cpanel'
```

After this MySQL Router has been started with the generated configuration

```
$ mysqlrouter -c /tmp/myrouter/mysqlrouter.conf&
```

10. Start Router continued

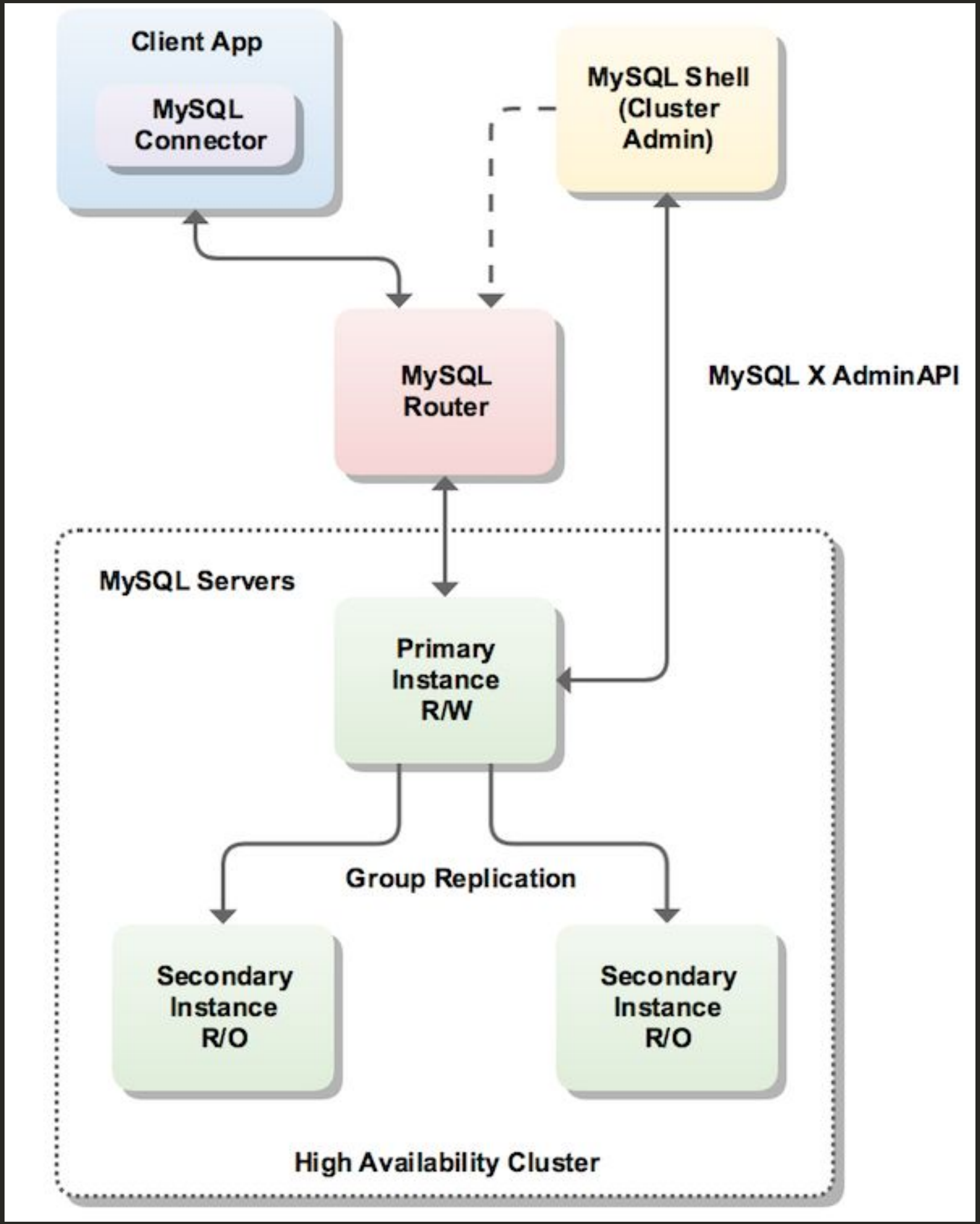
The cluster 'cpanel' can be reached by connecting to:

```
## MySQL Classic protocol              mysql -u root -p -P 6446
- Read/Write Connections: localhost:6446
- Read/Only Connections:  localhost:6447

## MySQL X protocol                    mysql -u root -p -P 64460
- Read/Write Connections: localhost:64460
- Read/Only Connections:  localhost:64470
```

Architecture

Place Router on the same platform as you application.

You application does not need to know what is behind the router and probably should not care. This way you can scale without changing application.

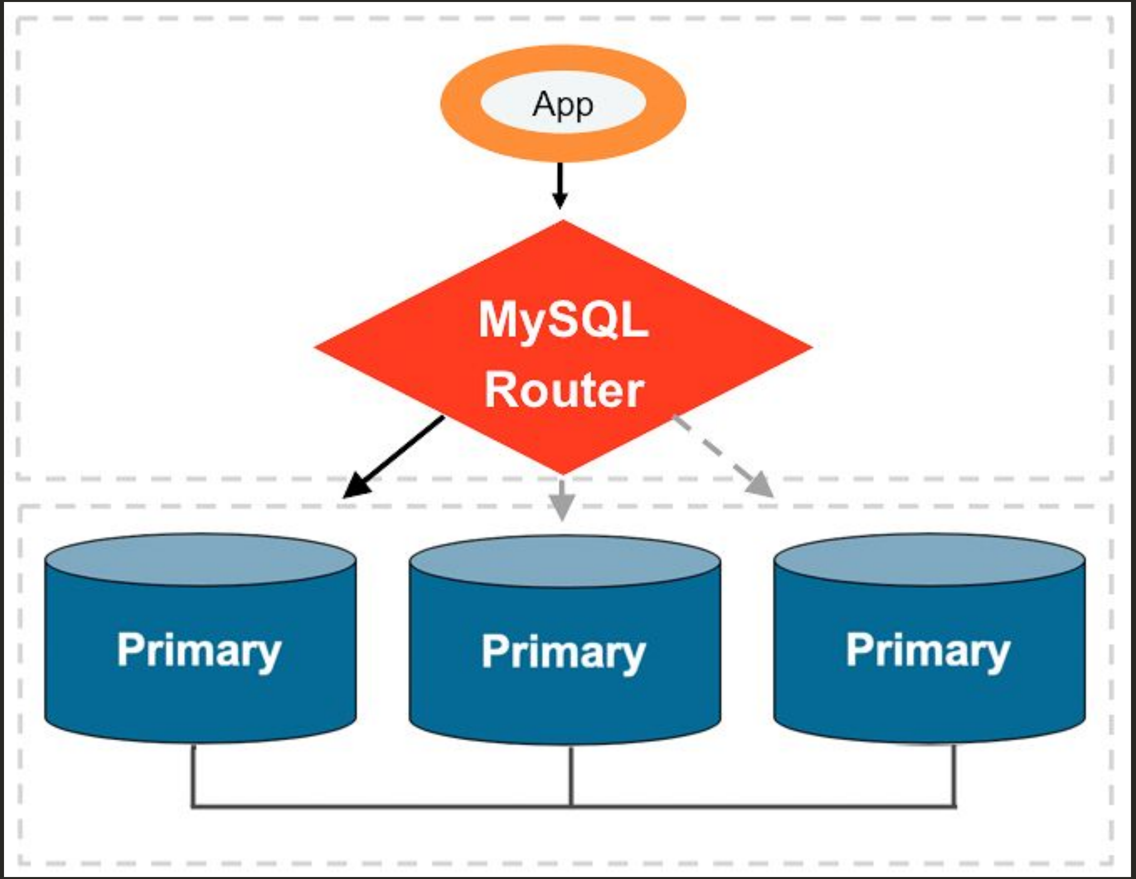


Want Multi-Primary?

```
JS > cluster.switchToMultiPrimaryMode()
Switching cluster 'cpanel' to Multi-Primary mode...

Instance '127.0.0.1:3330' was switched from SECONDARY to PRIMARY.
Instance '127.0.0.1:3310' remains PRIMARY.
Instance '127.0.0.1:3320' was switched from SECONDARY to PRIMARY.

The cluster successfully switched to Multi-Primary mode.
```

```
JS > cluster.status()
{
    "clusterName": "cpanel",
    "defaultReplicaSet": {
        "name": "default",
        "ssl": "REQUIRED",
        "status": "OK",
        "statusText": "Cluster is
ONLINE and can tolerate up to ONE
failure.",
        "topology": {
            "127.0.0.1:3310": {
                "address":
"127.0.0.1:3310",
                "mode": "R/W",
                "readReplicas": {},
                "replicationLag": null,
                "role": "HA",
                "status": "ONLINE",
                "version": "8.0.21"
            },
            "127.0.0.1:3320": {
                "address": "127.0.0.1:3320",
                "mode": "R/W",
                "readReplicas": {},
                "replicationLag": null,
                "role": "HA",
                "status": "ONLINE",
                "version": "8.0.21"
            },
            "127.0.0.1:3330": {
                "address": "127.0.0.1:3330",
                "mode": "R/W",
                "readReplicas": {},
                "replicationLag": null,
                "role": "HA",
                "status": "ONLINE",
                "version": "8.0.21"
            }
        },
        "topologyMode": "Multi-Primary"
    },
    "groupInformationSourceMember": "127.0.0.1:3310"
}
```

MySQL Router



Failover

Typically, a highly available MySQL setup consists of a single primary and multiple replicas and it is up to the application to handle failover, in case the MySQL primary becomes unavailable. Using MySQL Router, application connections will be transparently routed based on load balancing policy, without implementing custom application code.

Load Balancing

MySQL Router provides additional scalability and performance by distributing database connections across a pool of servers in a round-robin fashion.

Production InnoDb Cluster

Double check system configuration (run on all nodes)

- dba.configureLocalInstance("root@localhost:3306")

<u>Start the Cluster</u>

- var cluster = dba.createCluster('testCluster')
- cluster.addInstance('icadmin@ic-2:3306')
- cluster.addInstance('icadmin@ic-3:3306')
- mysqlrouter --bootstrap clusteradmin@db1:3306 --directory myrouter --user=root

Replica Set

InnoDB ReplicaSet a quick and easy way to get MySQL replication and MySQL Router up and running, making it well suited to scaling out reads, and provides manual failover capabilities in use cases that do not require the high availability offered by InnoDB cluster.

ReplicaSet automates a lot of the tasks found in traditional asynchronous replication.

Replica Set

An InnoDB ReplicaSet has several limitations compared to a InnoDB cluster and thus, it is recommended that you deploy InnoDB cluster wherever possible. Generally, a InnoDB ReplicaSet on its own does not provide high availability. Among the limitations of InnoDB ReplicaSet are:

- No automatic failover. In events where the primary becomes unavailable, a failover needs to be triggered manually using AdminAPI before any changes are possible again. However, secondary instances remain available for reads.
- No protection from partial data loss due to an unexpected halt or unavailability. Transactions that have not yet been applied by the time of the halt could become lost.
- No protection against inconsistencies after a crash or unavailability. If a failover promotes a secondary while the former primary is still available (for example due to a network partition), inconsistencies could be introduced because of the split-brain.

configure ReplicaSet

```
JS> dba.configureReplicaSetInstance()
JS> var rs = dba.createReplicaSet("example")
JS> rs.addInstance('rsadmin@host2')
```

configure ReplicaSet

```
JS> dba.configureReplicaSetInstance()
JS> var rs = dba.createReplicaSet("example")
JS> rs.addInstance('rsadmin@host2')
```

As an alternative to creating a new InnoDB ReplicaSet, you can also adopt an existing replication setup using the adoptFromAR option with dba.createReplicaSet().

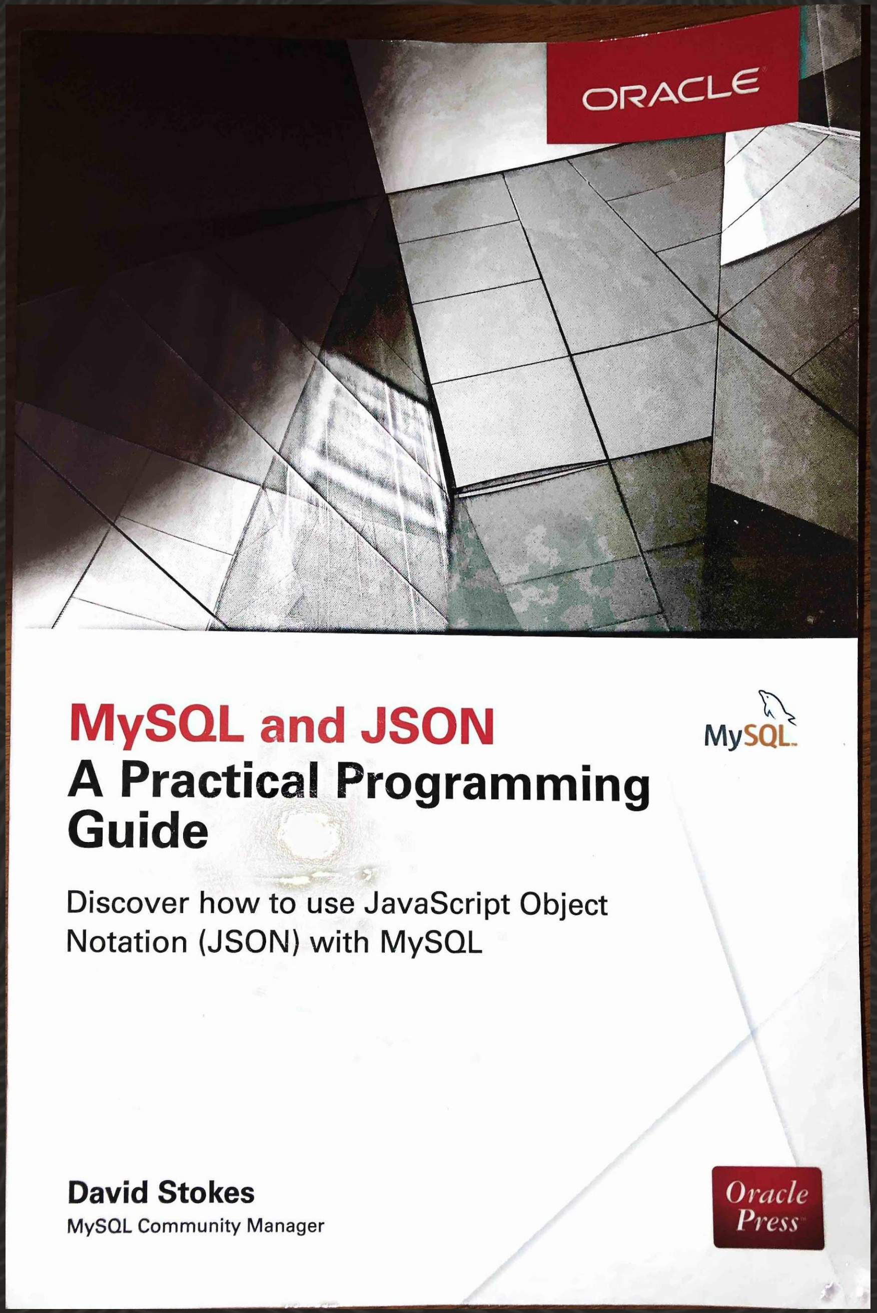
MySQL on the Internet

- MySQLCommunity.slack.com
- Forums.mysql.com
- MySQL.com & Planet.MySQL.com
- @MySQL
- LeFred.be -- My Colleague who blogs a lot on InnoDB Cluster

Thank You

Dave Stokes

Community Manager
MySQL Community Team
David.Stokes @ Oracle.com
@Stoker
ElephantDolphin. blogger.com
slideshare.net/davidmstokes
github.com/davidmstokes/cPanel


ORACLE

MySQL and JSON
A Practical Programming Guide

Discover how to use JavaScript Object Notation (JSON) with MySQL

David Stokes
MySQL Community Manager

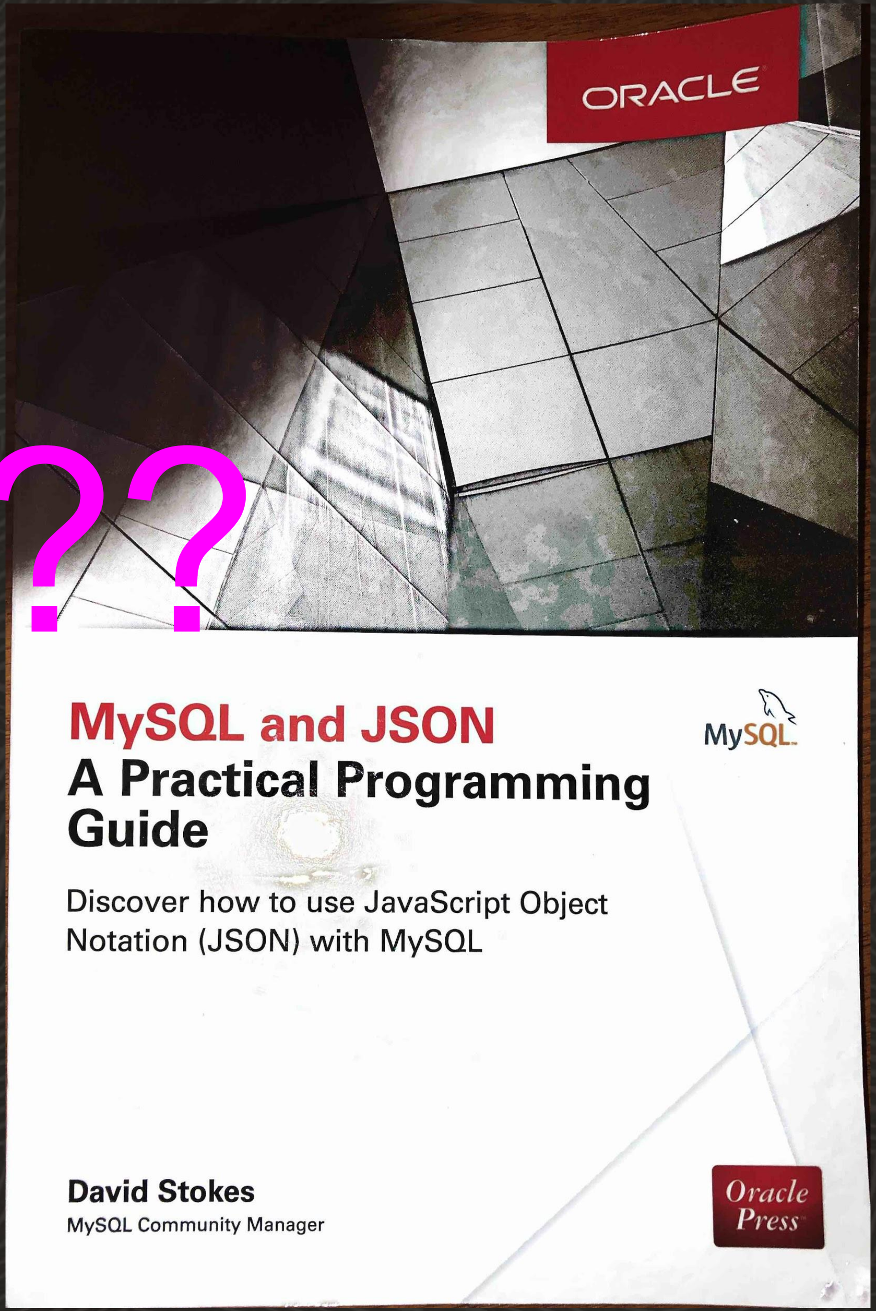
Oracle Press

Thank You

Q&A??

Dave Stokes

Community Manager
MySQL Community Team
David.Stokes @ Oracle.com
@Stoker
ElephantDolphin. blogger.com



ORACLE

MySQL and JSON
A Practical Programming Guide

Discover how to use JavaScript Object Notation (JSON) with MySQL

David Stokes
MySQL Community Manager

Oracle Press