

Migrating to Aurora MySQL and Monitoring with PMM

Percona Technical Webinars August 1, 2018



Introductions

Introduction



Vineet Khanna (Autodesk)
Senior Database Engineer
vineet.khanna@autodesk.com



Tate McDaniel (Percona)
Senior MySQL DBA
tate.mcdaniel@percona.com

This talk's agenda

Agenda



What is this talk about?

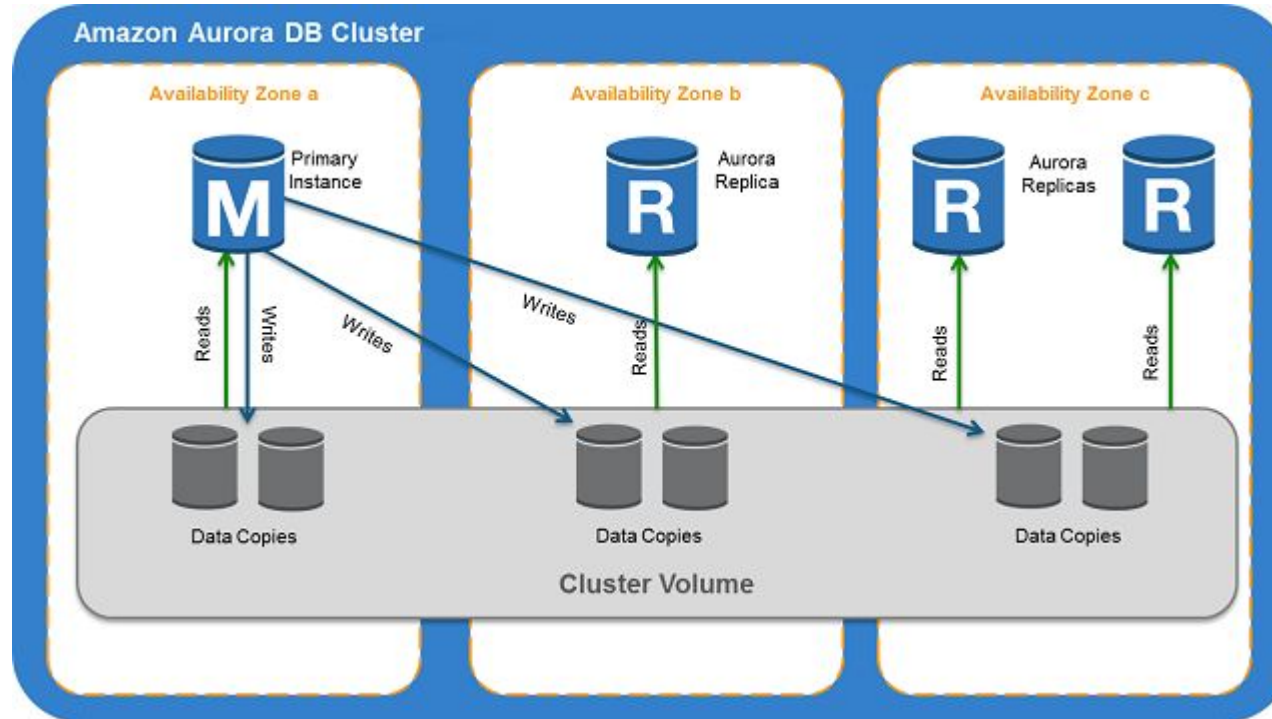
- A real life migration from EC2 backed MySQL instances to managed Aurora clusters
 - How do I determine if Aurora is the correct solution for my application?
 - What does the decision making process look like?
 - What does a POC look like?
- Implementing code reusable version and infrastructure tracking
 - Adapting Terraform to AWS Aurora
 - Open source development of tools for devops - code reusable
 - Auditing changes to the database infrastructure.
- Monitoring and evaluating the effectiveness of a major infrastructure change
 - Using PMM and other tools to monitor the new environment
 - Comparison of old and new environments
 - Justification of the infrastructure shift



In this talk:
Aurora = AWS Aurora MySQL 5.6

AWS Aurora - What?

AWS Aurora MySQL Architecture



AWS Aurora - Why?

Why Aurora? Pros...

General benefits of AWS Aurora MySQL:

- Replication Lag less than 100 ms within same region
- 6 copies of your data across 3 Availability Zones
- Scale storage size automatically to 64 TB
- Supports up to 15 Replicas
- Isolates the database buffer cache from database processes
- Does not require crash recovery replay of database redo logs
- Multi-AZ is built-in and automatic
- Continuous backup to S3
- Allows storage encryption
- Zero Downtime Patching available without binlog
- Fast Database Cloning
- Auto Scaling - Read Replica

Why Aurora? Cons...

General downsides of AWS Aurora MySQL:

- Cross-region replica based on binlog
- Table level corruption suffers across the cluster
- Point in Time Recovery not possible
 - However AWS provides Point in Time Restore
- Delayed slave not possible to survive drop table disaster
- No change buffering
- Stick to MySQL version 5.6.10 & 5.7.12
- Performance Schema disabled for Aurora MySQL 5.7
- External Plugins not supported
- Supports only InnoDB storage engine
- Closed Source

MySQL RDS vs Aurora Performance Test

Machines:

- **Client Machine:** Instance Type: m4.4xlarge (CPU 16, Mem. 60G)
- **Database Machine:** Instance Class: db.r3.8xlarge (CPU 32, Mem. 244G)
 - Master & Read Replica

Benchmarking Tool:

- Sysbench 1.0
 - Tables: 10 tables x 250 million rows (around 50G each table)

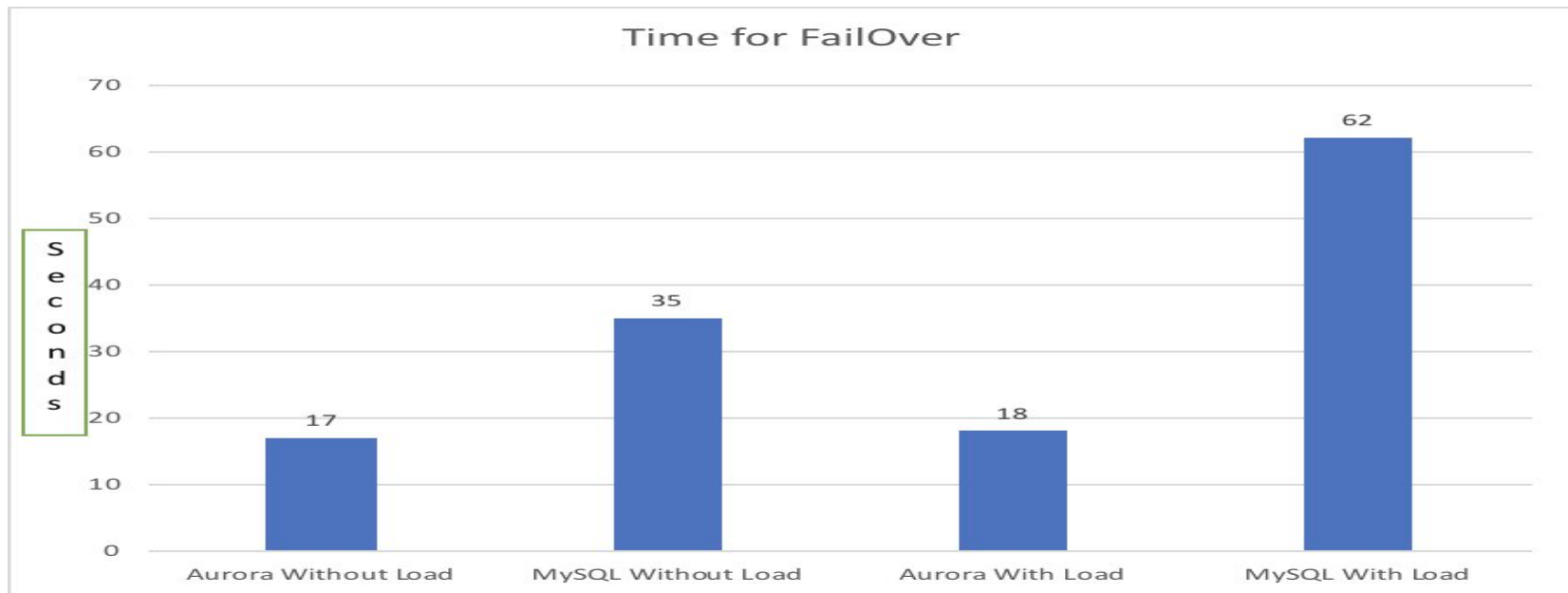
Benchmark Test:

- OLTP RW/RO
- Failover Test
- Latency(Slave Lag) Test

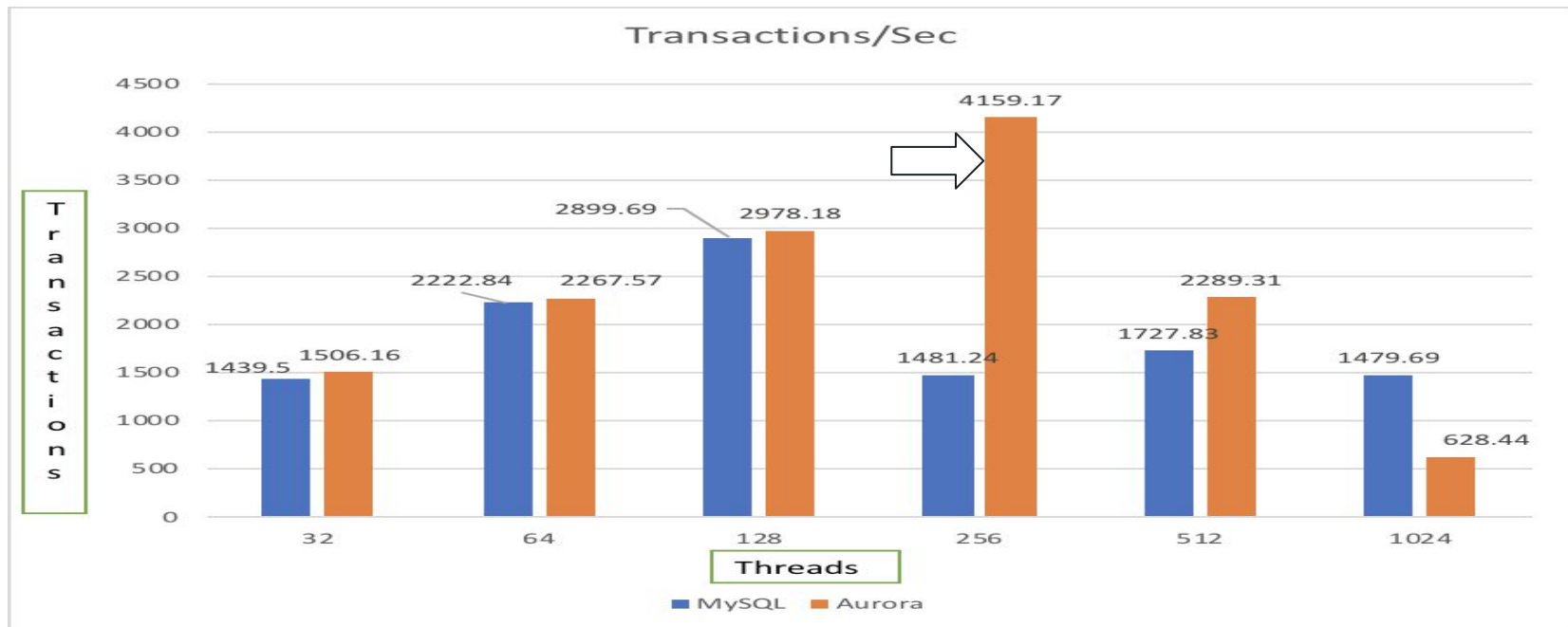
Test Duration:

- 60 min each test (OLTP)

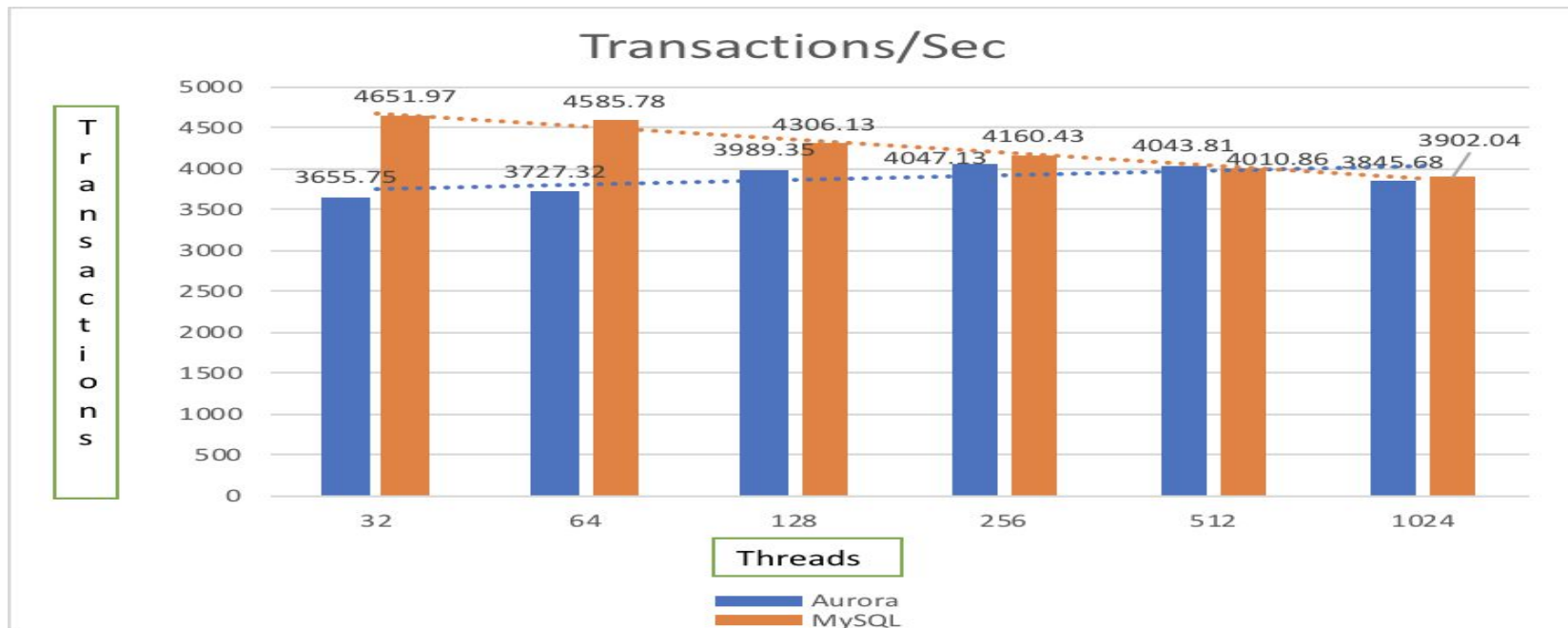
MySQL RDS vs Aurora Failover Test



MySQL RDS vs Aurora OLTP RW Test

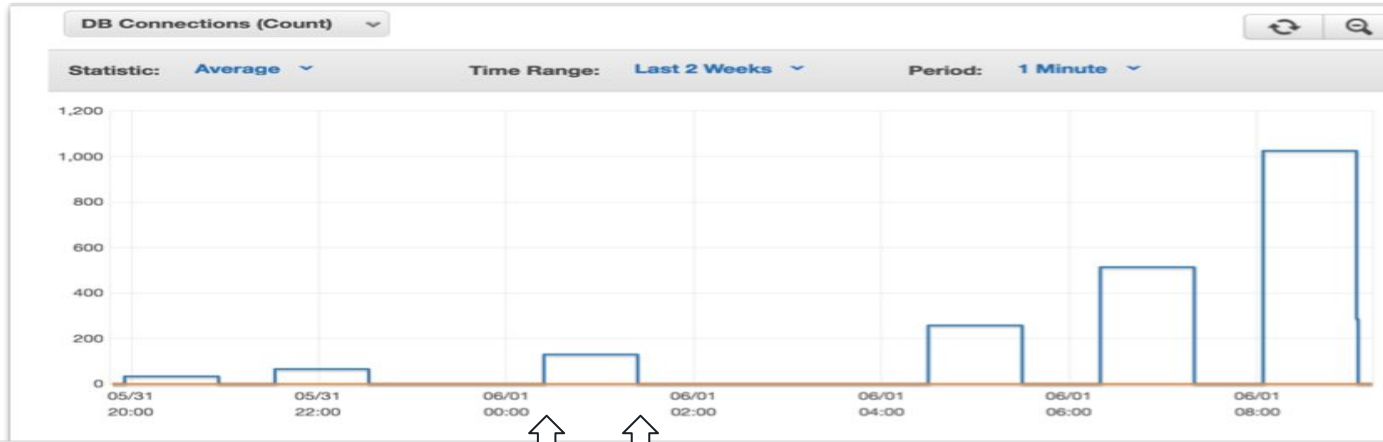



MySQL RDS vs Aurora OLTP RO Test



MySQL RDS Latency Test





CloudWatch Alarms:  No alarms configured [Create Alarm](#)

rds-poc | rds-poc-replica

MySQL Aurora Latency Test



DB Connections (Count)

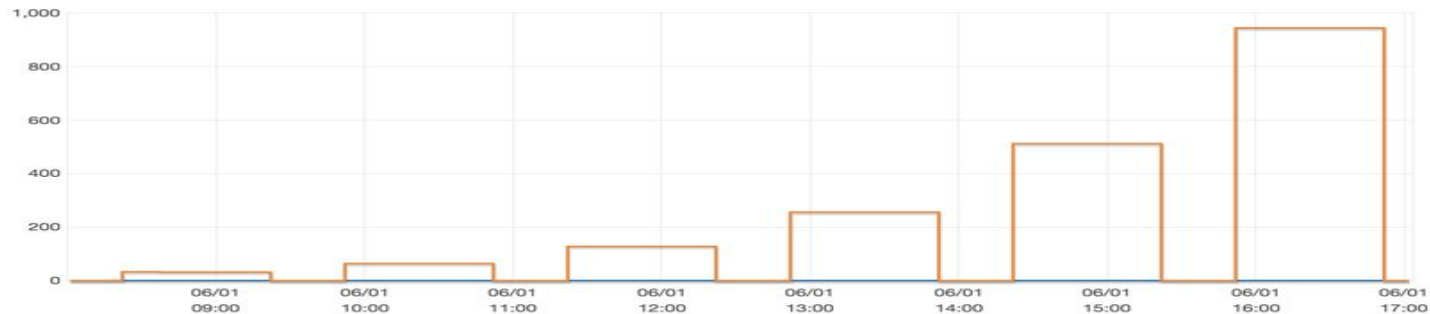
DB Connections (Count)



Statistic: **Average**

Time Range: **Last 2 Weeks**

Period: **1 Minute**



DB Connections (Count)

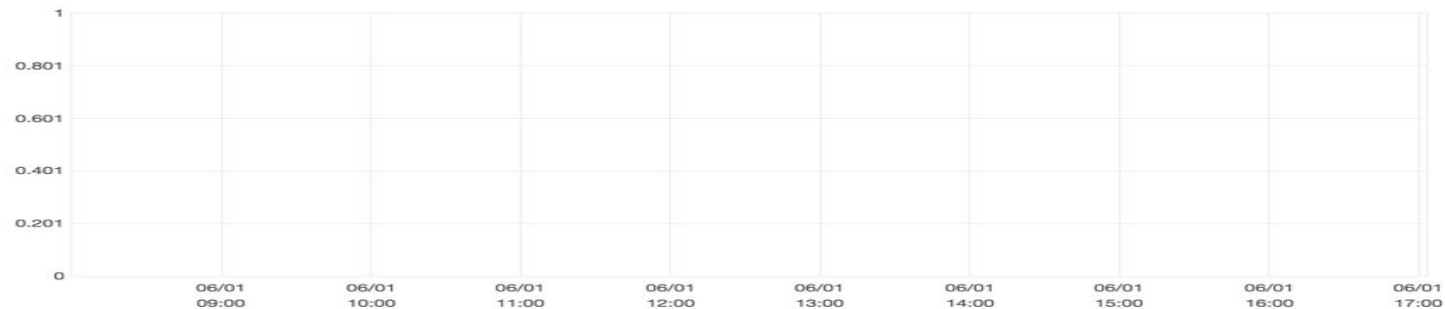
Replica Lag (ms)



Statistic: **Average**

Time Range: **Last 2 Weeks**

Period: **1 Minute**



CloudWatch Alarms: No alarms configured

Create Alarm

aurora-poc-slave | aurora-poc



The Future of Aurora

The Future of Aurora

Upcoming features to be rolled out in Q2 or beyond:

- Multi Master Multi Region AWS Aurora
- AWS Serverless Aurora
- Parallel Query
- Database backtrack

Verify Schema/Data Consistency

Verify Schema Consistency

Why we need to verify Schema Consistency

- Functions are not imported automatically
- Stored procedures are not imported automatically.
- User accounts are not imported automatically.
- Views with definer root@localhost are not accessible

Tools & Commands

- MySQL Utilities: mysqldiff

```
mysqldiff --difftype=sql --changes-for=server2 --server1=$USER:$PASS@$EC2IP  
--server2=$USER:$PASS@$AURORAIP $DBNAME:$DBNAME
```

- MySQL Utilities: mysqldbcompare

```
mysqldbcompare --difftype=sql --server1=$USER:$PASS@$EC2IP  
--server2=$USER:$PASS@$AURORAIP $DBNAME:$DBNAME --changes-for=server2
```

Verify Data Consistency

Verify Data Consistency Using pt-table-checksum

- If you are running MySQL on STATEMENT binlog format
- Run pt-table-checksum directly from EC2 based MySQL Master
- pt-table-checksum

```
pt-table-checksum --recursion-method dsn=h=localhost,D=percona,t=dsns  
--nocheck-replication-filters --ignore-databases  
mysql,performance_schema,information_schema > checksum.log 2>&1
```

- pt-table-sync

```
pt-table-sync --replicate percona.checksums --sync-to-master  
192.168.1.245 --user restore --pass 'PASSWORD' --print --verbose
```

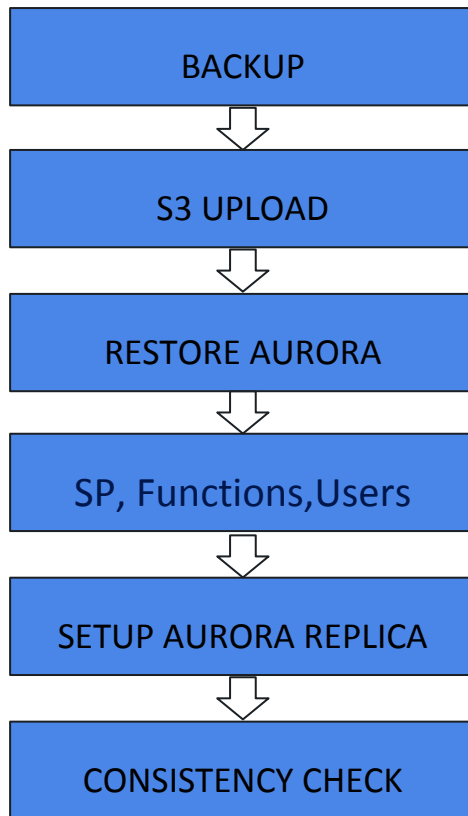
Verify Data Consistency Using Paused Slave

- Create Aurora Cluster as a Slave of Slave
- Pause Slave and checksum table

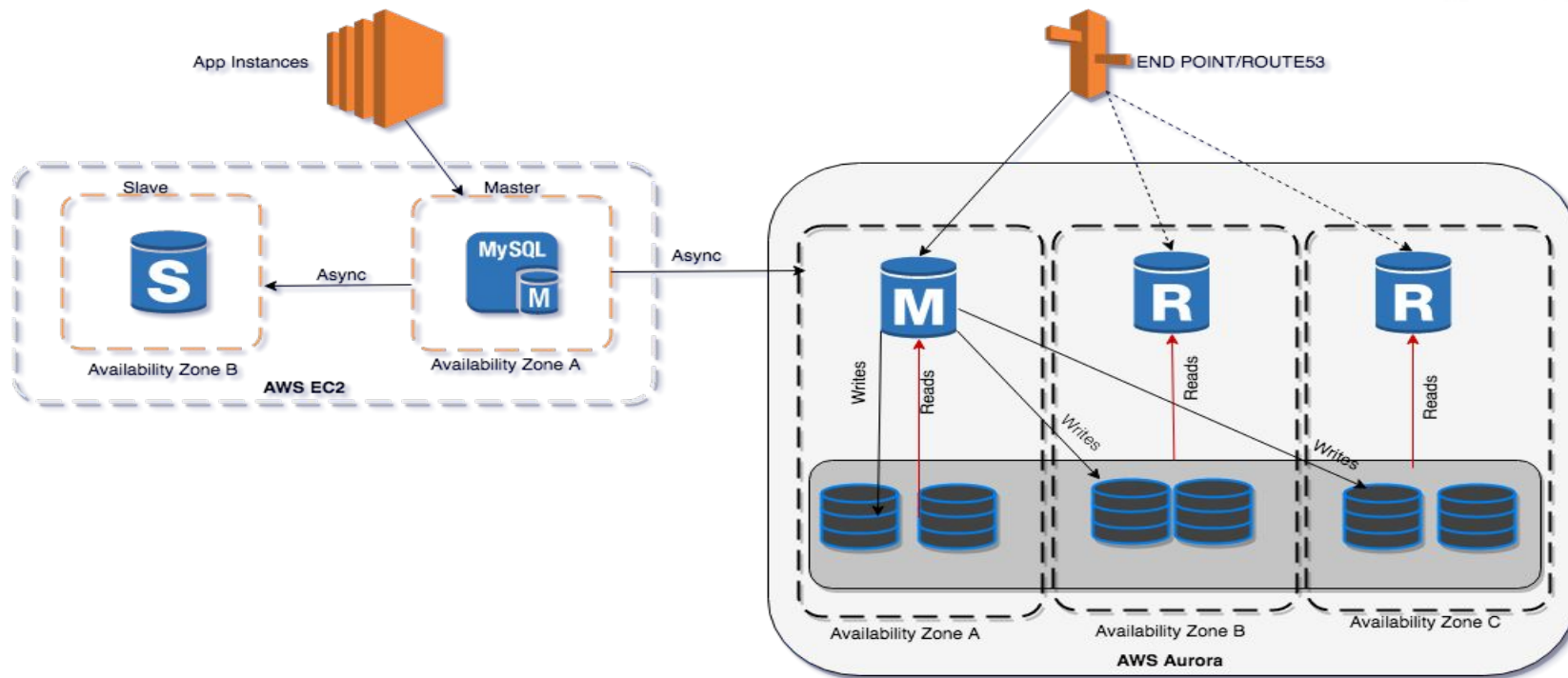
Aurora Migration

1:1 Migration From EC2 MySQL to Aurora

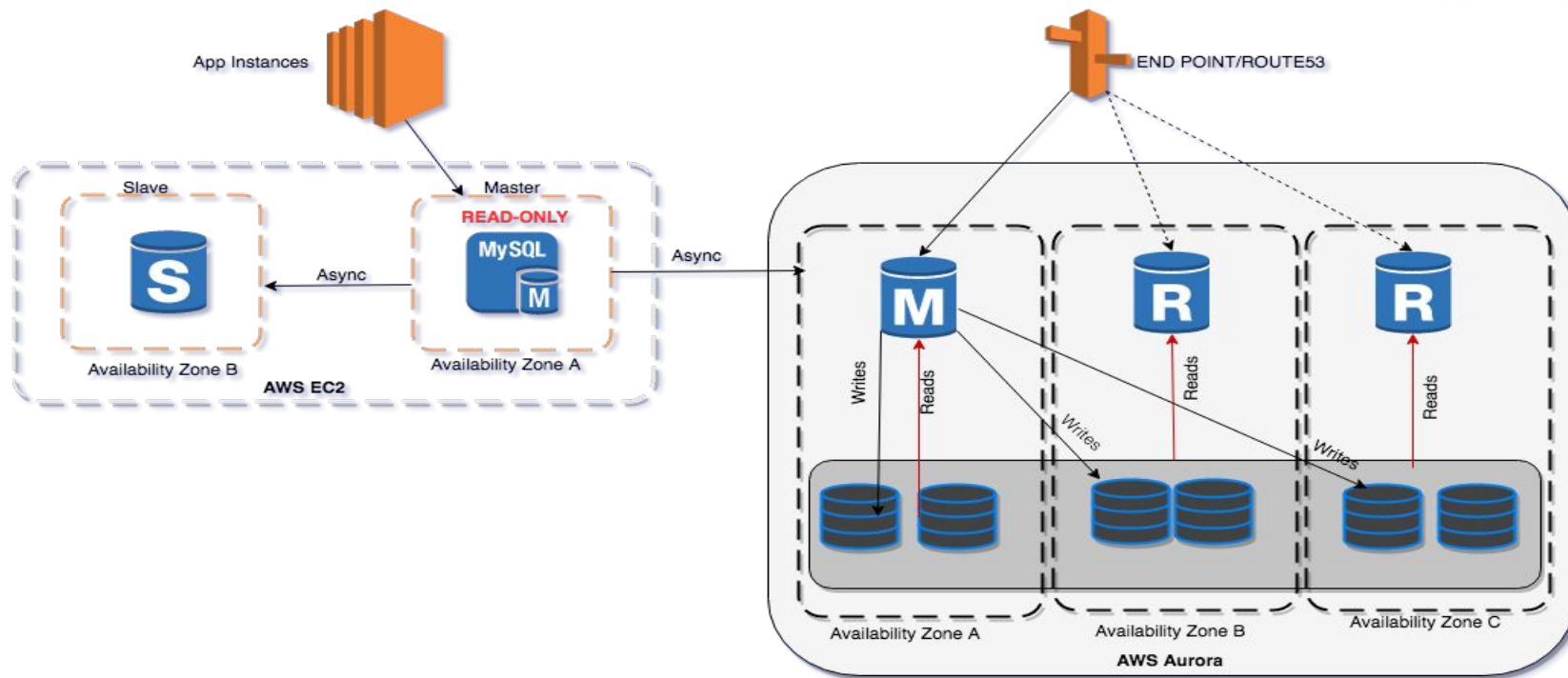
1:1 Migration: EC2 MySQL to Aurora



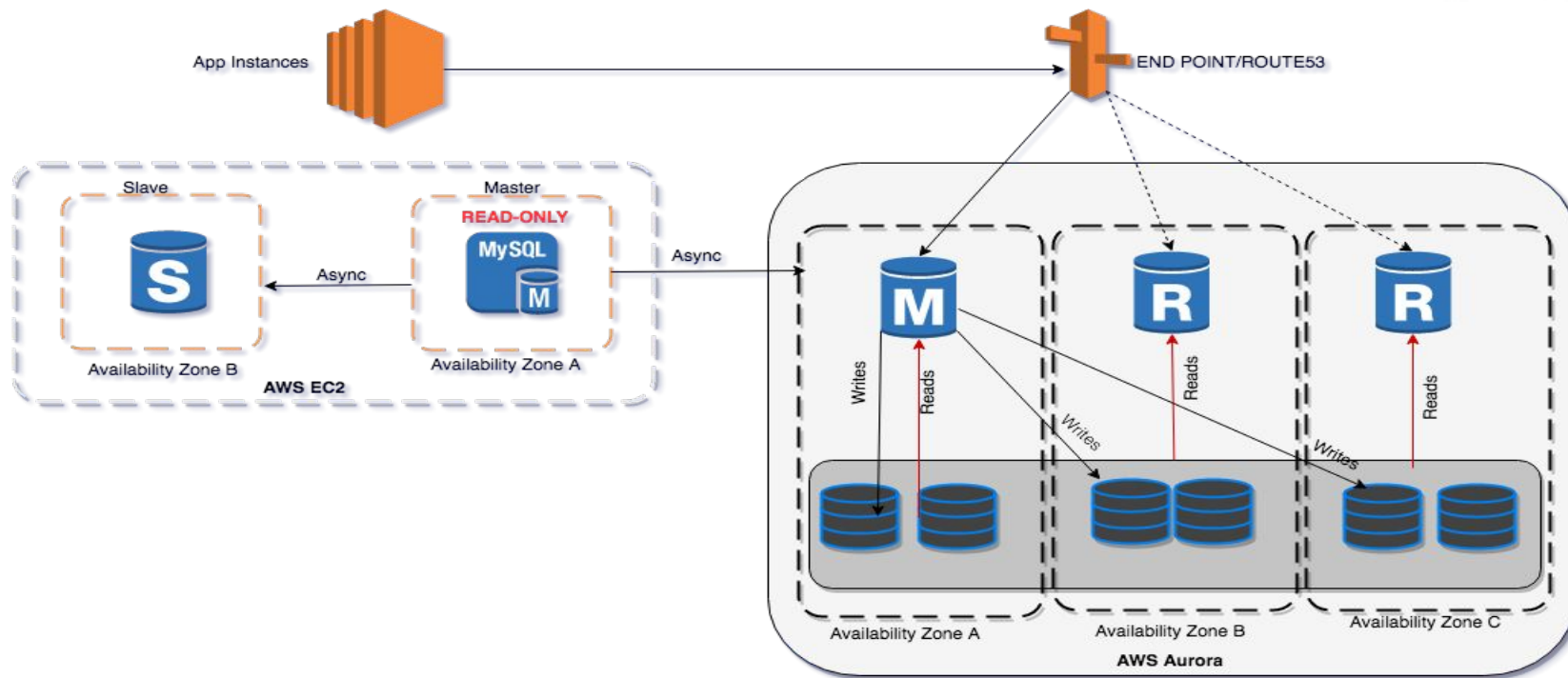
1:1 Migration: EC2 MySQL to Aurora



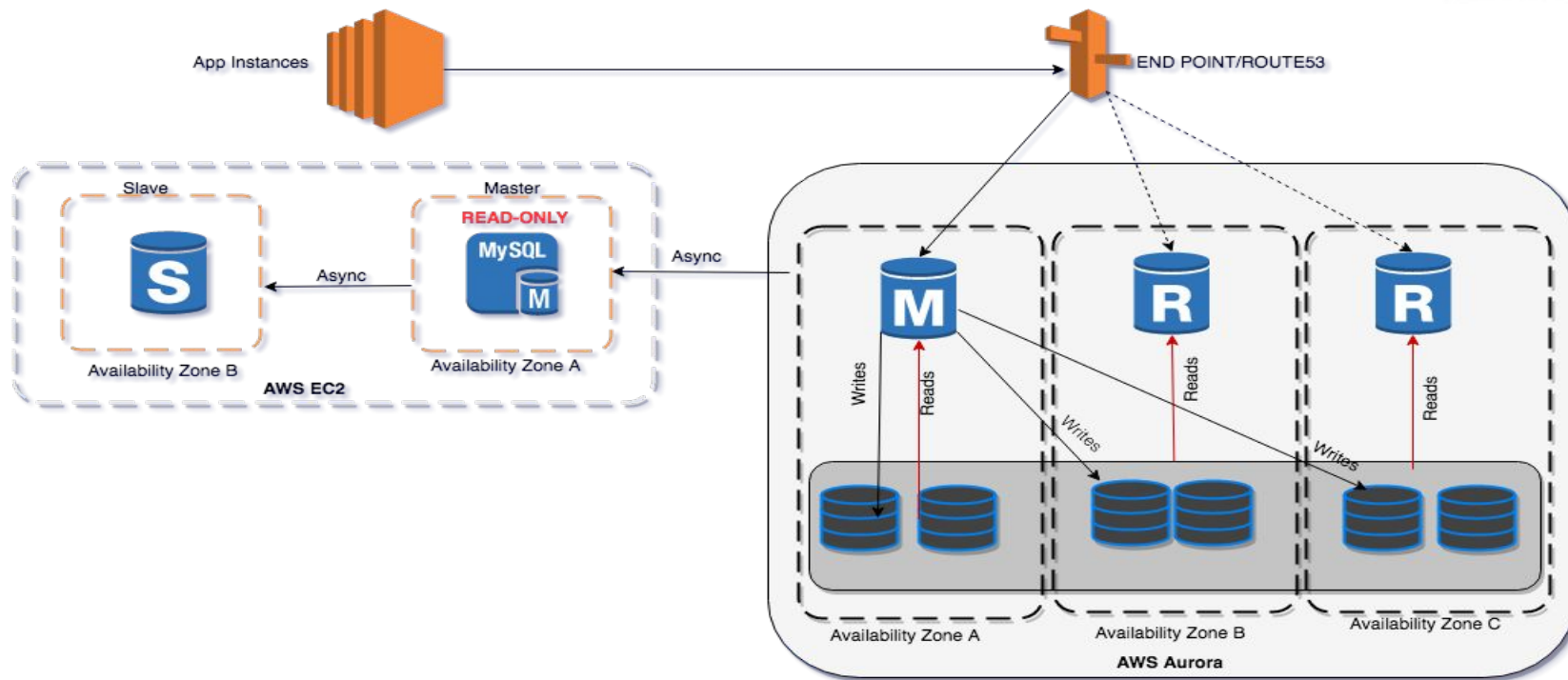
1:1 Migration: EC2 MySQL to Aurora



1:1 Migration: EC2 MySQL to Aurora



1:1 Migration: EC2 MySQL to Aurora



1:1 Migration: EC2 MySQL to Aurora

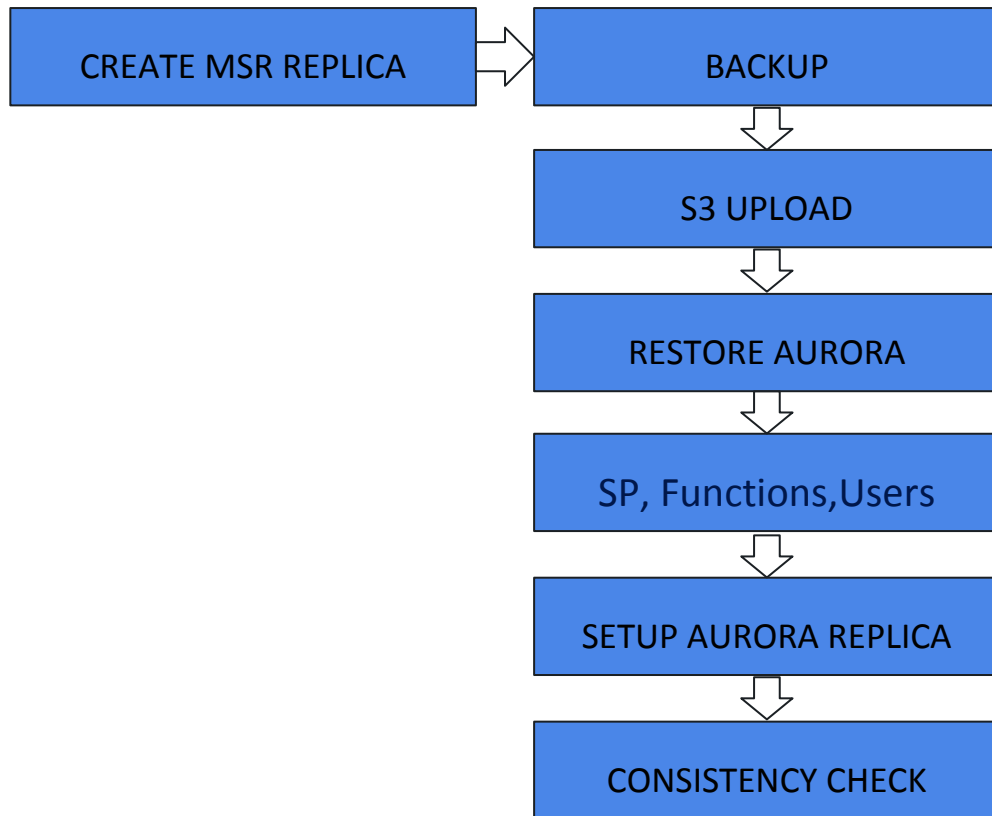


Migrating EC2 Based MySQL to Aurora MySQL:

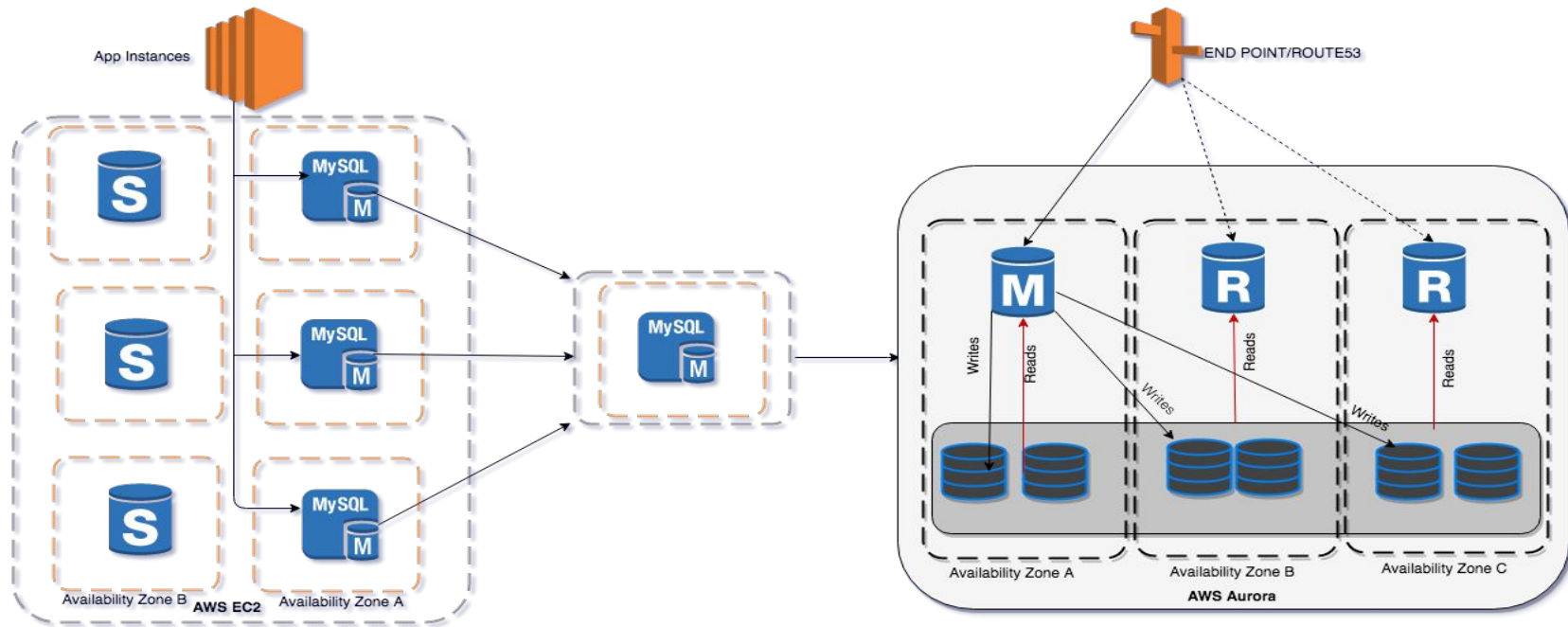
1. Create Backup using Percona XtraBackup
2. Upload to S3 bucket
3. Restore Aurora Cluster using S3 backup
4. Manually create Stored Procedure, Functions & Localhost Users
5. Make Aurora as a replica of EC2 based MySQL
6. Wait until the replication catch-up
7. Run Schema & Data Consistency checks
8. Update ODBC/JDBC connection string to point Aurora
9. Enable Read Only on EC2 based MySQL Master
10. Capture Aurora binlog position
11. Reload Config/Restart Web Servers (Ideally this should take less 1 min to avoid any downtime)
12. Setup Reverse Replication for Failback

Many:1 Migration From EC2 MySQL to Aurora Using MSR

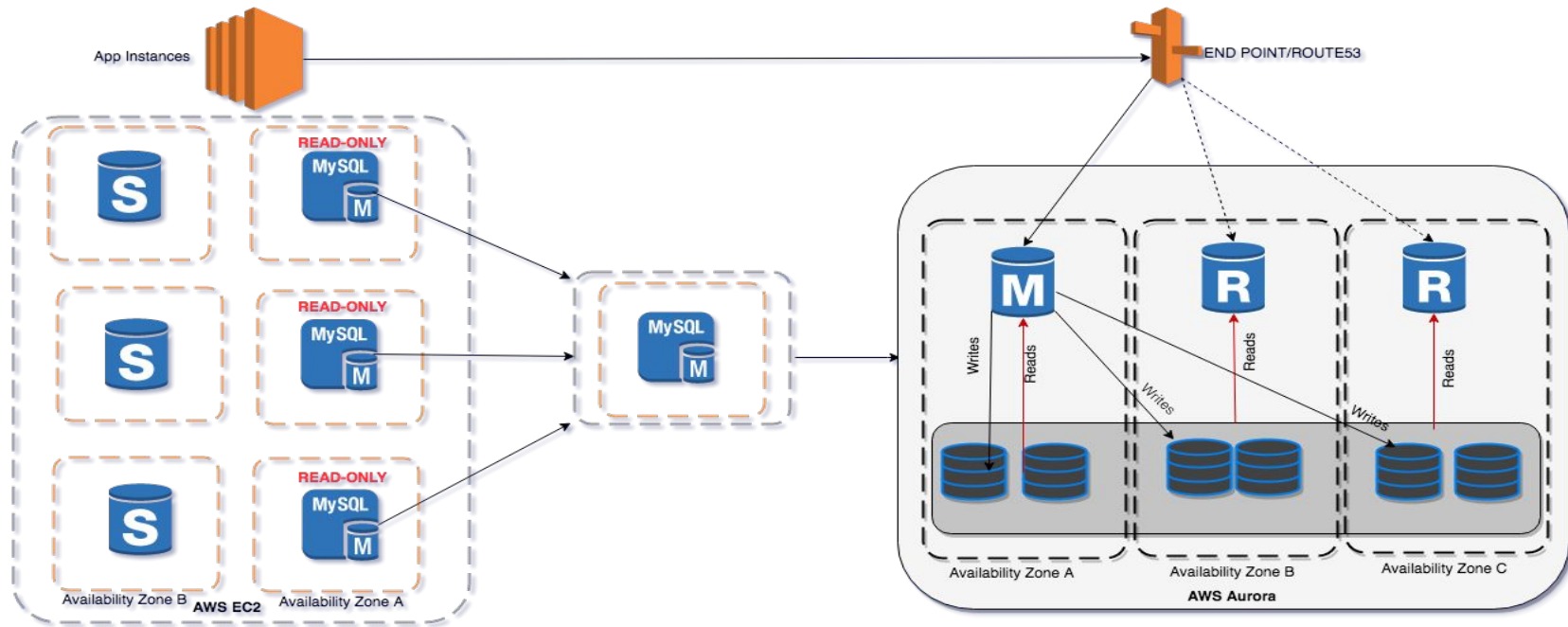
Many:1 Migration: EC2 MySQL to Aurora Using MSR



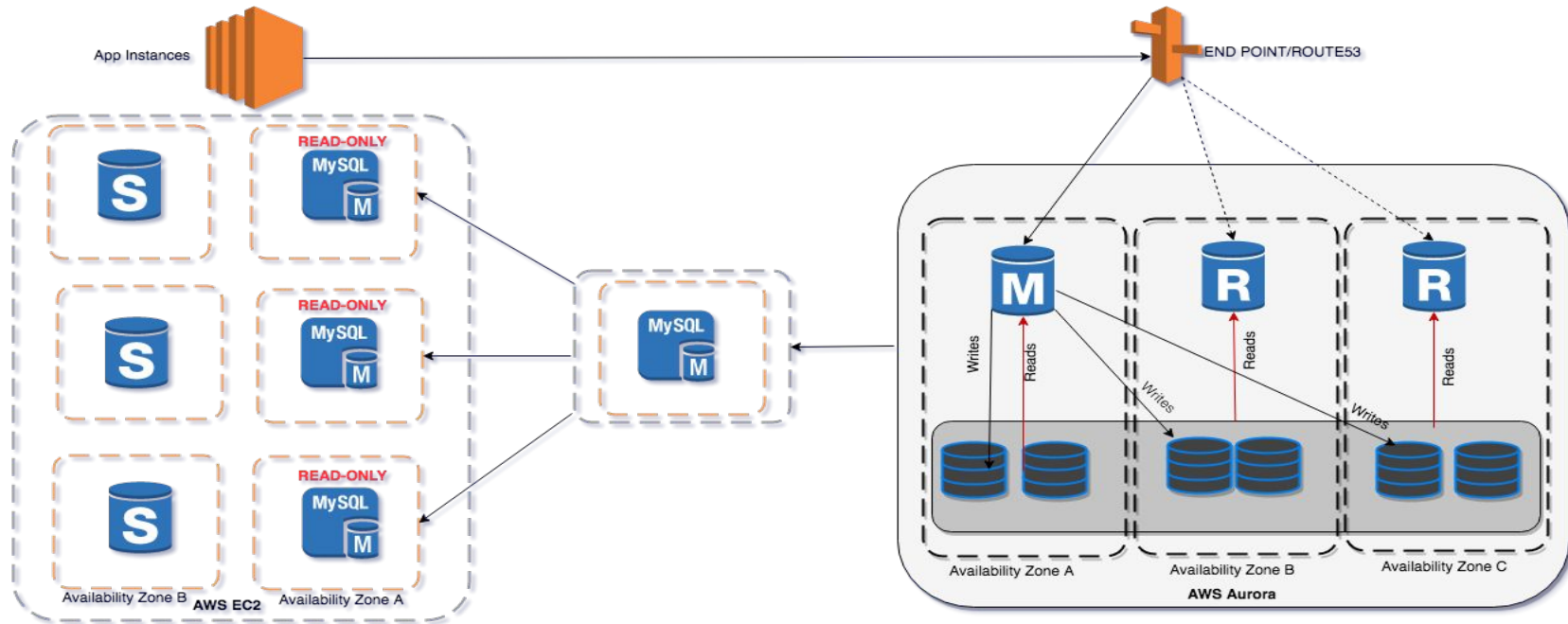
Many:1 Migration: EC2 MySQL to Aurora Using MSR



Many:1 Migration: EC2 MySQL to Aurora Using MSR



Many:1 Migration: EC2 MySQL to Aurora Using MSR



Many:1 Migration: EC2 MySQL to Aurora Using MSR



Migrating & Consolidate EC2 Based MySQL to Aurora Using Multi Source Replication:

1. Create Backups of All MySQL Servers using mysqldump
2. Create EC2 instance with Percona 5.7 to use Multi Source Replication Feature
3. Dump data from multiple sources to newly created EC2 instance & configure replication
4. Create Backup using Percona XtraBackup
5. Upload to S3 bucket
6. Restore Aurora Cluster using S3 backup
7. Manually create Stored Procedure, Functions & Localhost Users
8. Make Aurora as a replica of EC2 based MySQL
9. Wait until the replication catch-up
10. Run Schema & Data Consistency checks

Many:1 Migration: EC2 MySQL to Aurora Using MSR

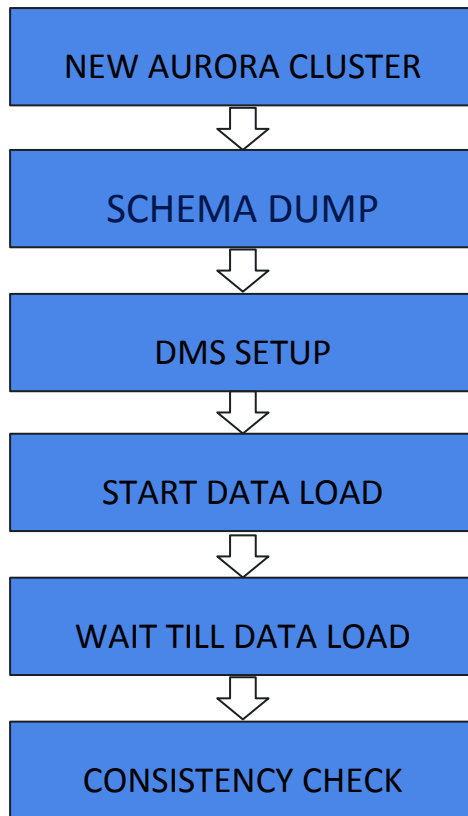


Migrating & Consolidate EC2 Based MySQL to Aurora Using Multi Source Replication:

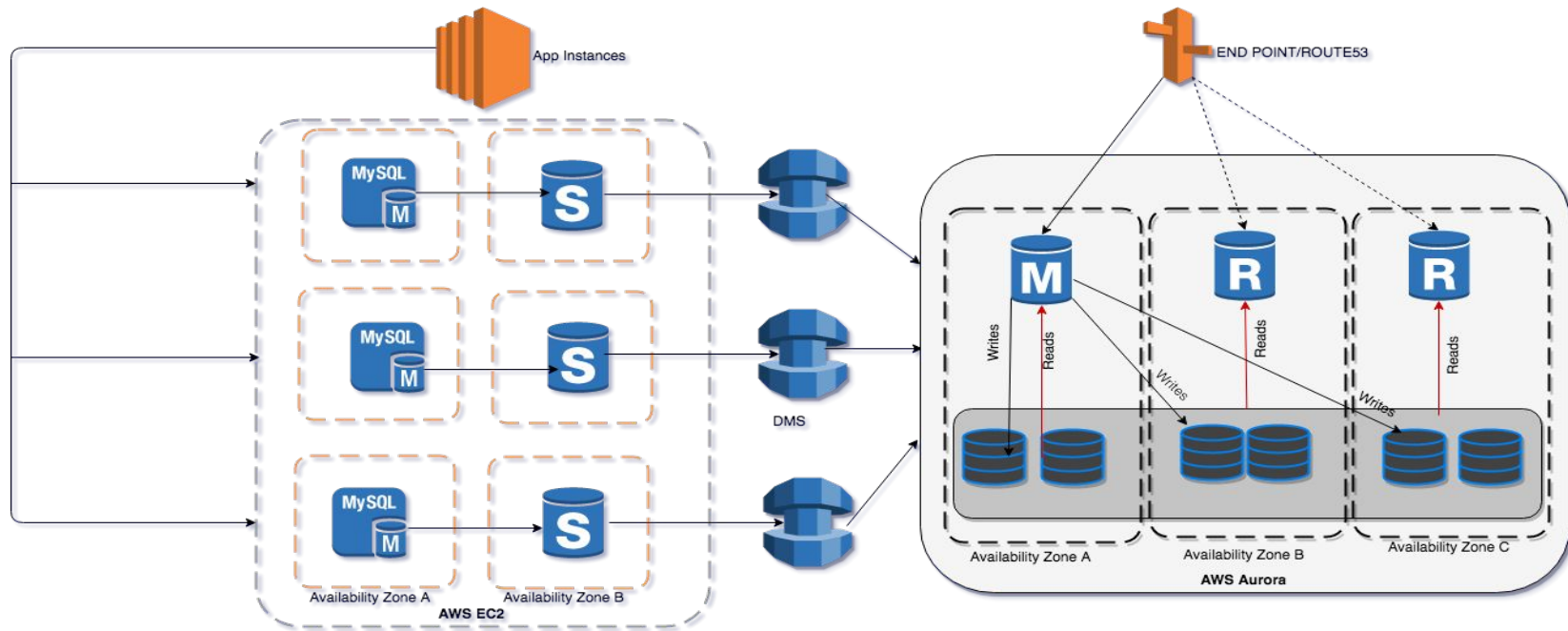
- 11 Update ODBC/JDBC connection string to point Aurora
- 12 Enable Read Only on MySQL Masters
- 13 Capture Aurora binlog position
- 14 Reload Config/Restart Web Servers (Ideally this should take less 1 min to avoid any downtime)
- 15 Setup Reverse Replication for Failback

Many:1 Migration From EC2 MySQL to Aurora Using DMS

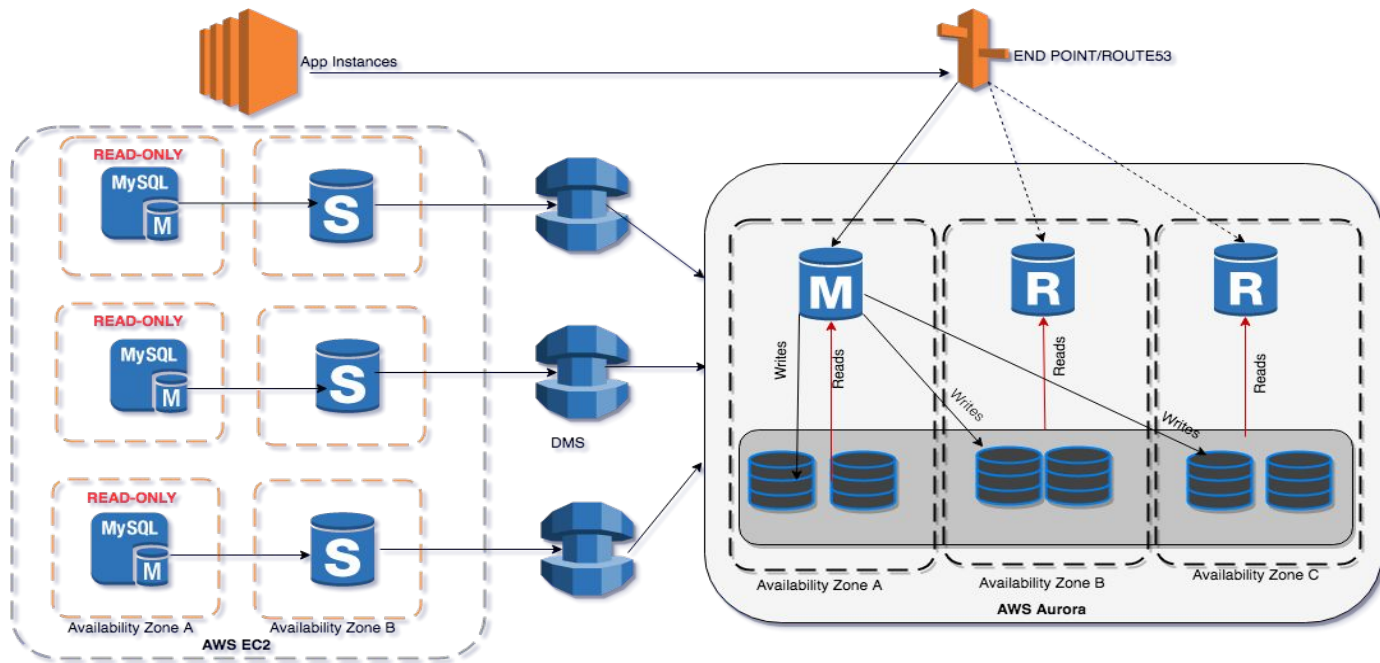
Many:1 Migration: EC2 MySQL to Aurora Using AWS DMS



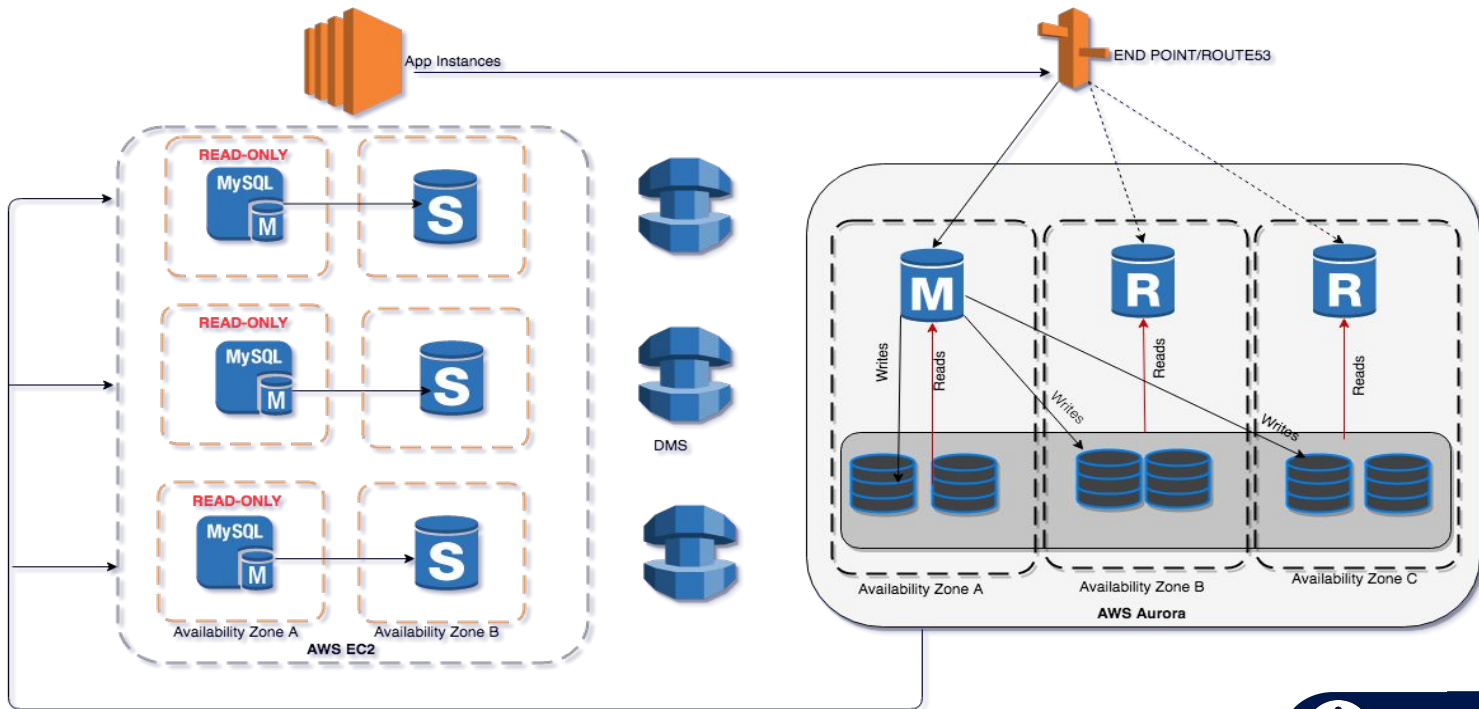
Many:1 Migration: EC2 MySQL to Aurora Using AWS DMS



Many:1 Migration: EC2 MySQL to Aurora Using AWS DMS



Many:1 Migration: EC2 MySQL to Aurora Using AWS DMS



Many:1 Migration: EC2 MySQL to Aurora Using AWS DMS



Migrating & Consolidate EC2 Based MySQL to Aurora Using DMS:

1. Create Empty AWS Aurora Cluster
2. Take dump of Schema, Stored Procedure, Functions & Users and Transfer manually to Aurora
3. Create AWS DMS for each MySQL source to target Aurora Cluster
 - a. AWS DMS will take care of initial load and ongoing replication changes
4. Wait until the initial load complete and replication catch-up
5. Run Schema & Data Consistency checks
 - a. Can't utilize pt-table-checksum/sync
6. Update ODBC/JDBC connection string to point Aurora
7. Enable Read Only on MySQL Masters
8. Capture Aurora binlog position
9. Reload Config/Restart Web Servers (Ideally this should take less 1 min to avoid any downtime)
10. Setup Reverse Replication for Failback

MSR vs DMS Migration Comparison



Comparison Points	MSR	DMS
Schema Migration	Automatic (S3 restore)	Manual
Replication from Master	Yes	No
Data Consistency Check	Yes	Yes with limitations
Online DCC	Yes	No
Performance (Full LOB)	Doesn't Matter	Very Slow
Seconds Behind Master	Yes	CDCLatencySource/Target
Setup	Complex	Simple
Binlog Format	All	ROW

PMM dashboards

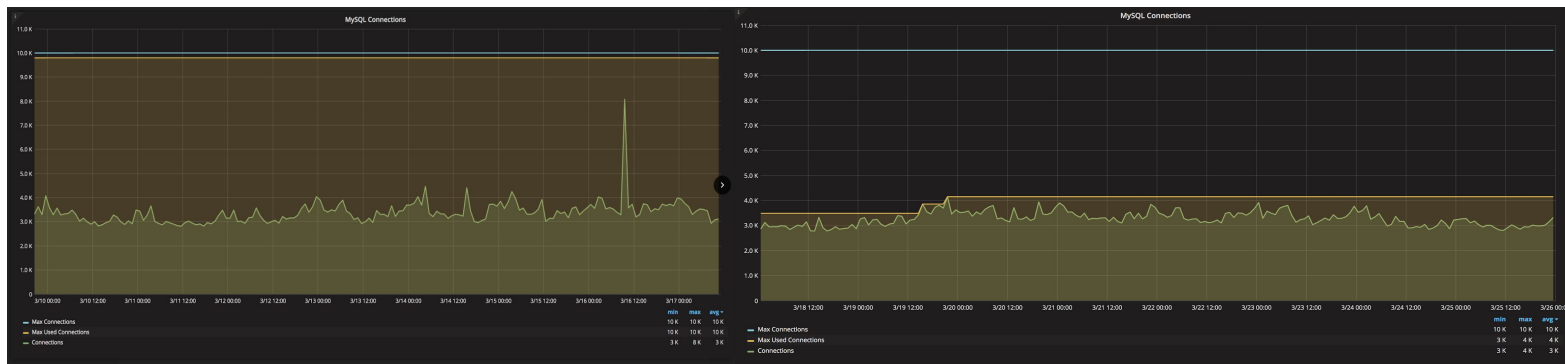
MySQL Connections



EC2 on left, Aurora on right

Consistent number of connections

Neither technology is pushed anywhere near the limit

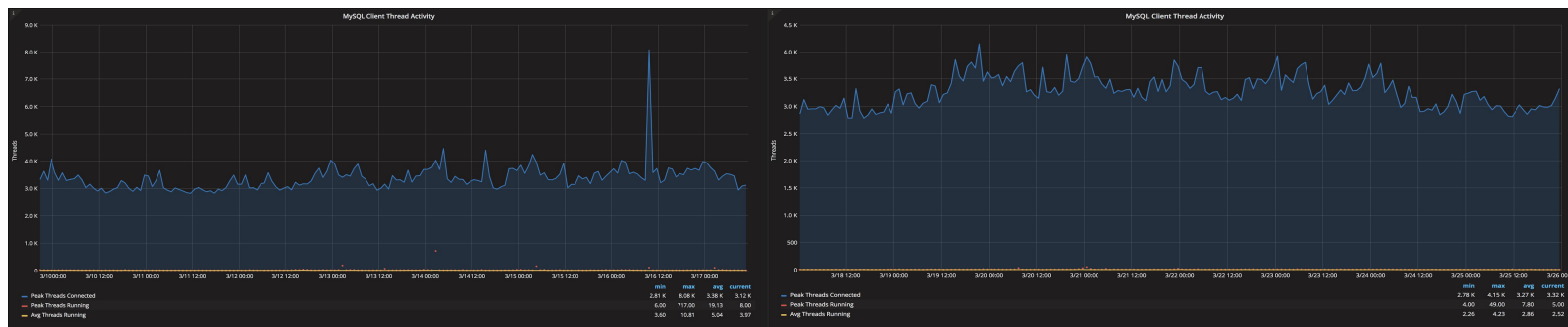


MySQL Client Thread Activity



The most interesting metric is threads running

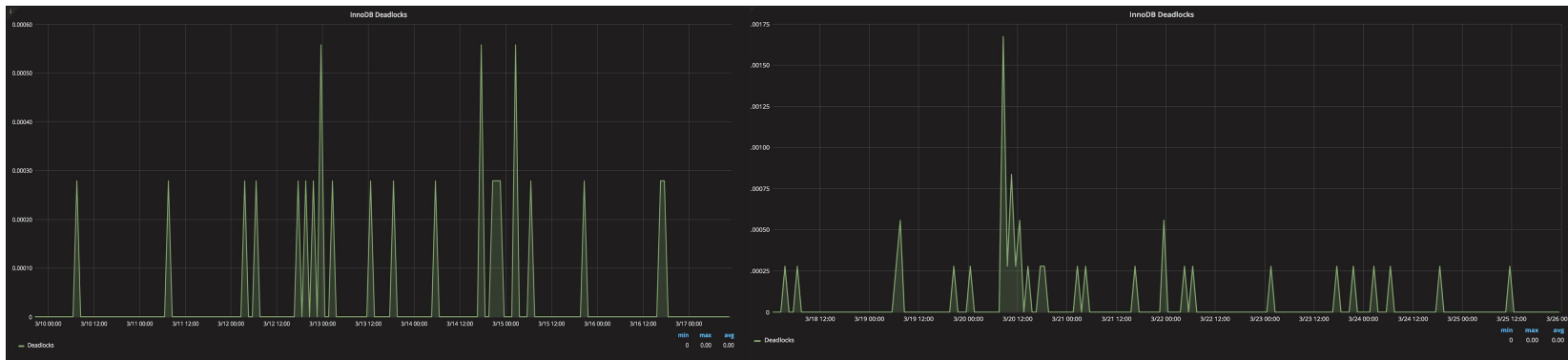
In both cases this metric is low and in both cases the threads connected are similar



InnoDB Deadlocks



Deadlocking is symptom of poor management of concurrency
Both technologies have a low number of deadlocks



InnoDB Row Lock Time



Row contention metric

Aurora has a slightly avg row lock time

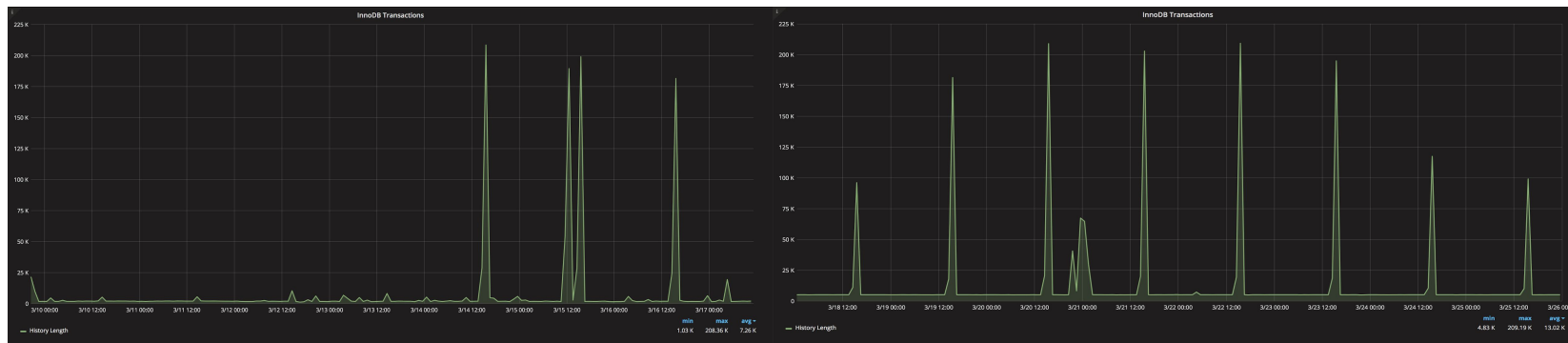
Pattern is consistent



InnoDB Transactions



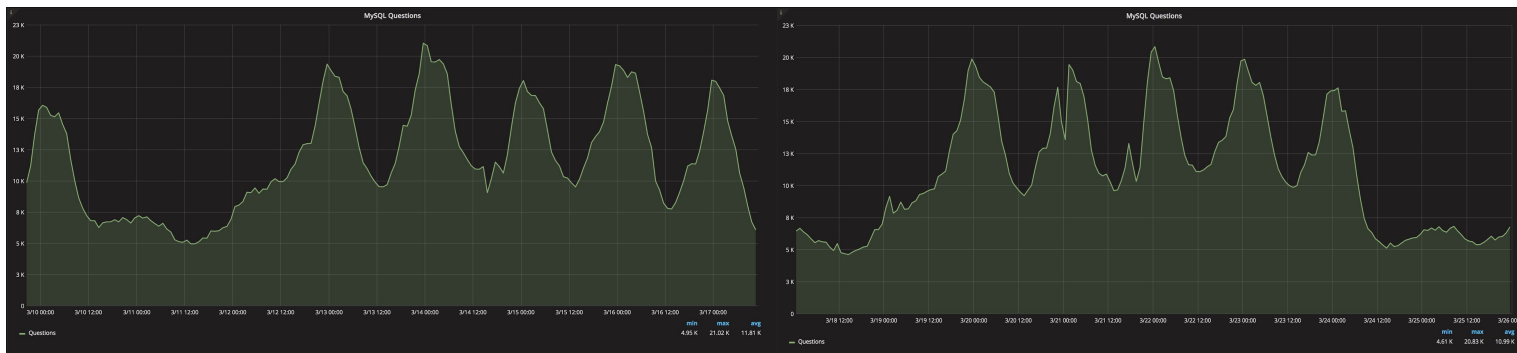
InnoDB transactions give us a glimpse into average and peak loads
Both technologies achieved similar peak loads



MySQL Questions



Questions are similar to transactions but represent actual activity in innodb at any given time
Consistent peaks and valleys indicate a similar performance profile



Build Aurora Infra

Using Terraform



Why Terraform:

- Infrastructure as Code
- Open Source
- Supports multiple Cloud providers
- Reusable code (Dev → Stage → Prod)

Problem we faced during Terraform Implementation:

- Terraform doesn't support Aurora Restore using S3

Our Implementation:

- Still we want to have Infrastructure as Code
- Terraform supports instance creation separately
- We wrote our own python code to launch Aurora Cluster
- We used XtraBackup to restore Aurora Cluster

RDS Aurora: Add feature to Restore Aurora Cluster from S3

New issue

#2440



hashibot opened this issue on 27 Nov 2017 · 3 comments



hashibot commented on 27 Nov 2017

Assignees

No one assigned

This issue was originally opened by @khannavin as [hashicorp/terraform#16736](#). It was migrated here as a result of the [provider split](#). The original body of the issue is below.

Labels

new-resource

service/rds

Terraform Version

Terraform v0.10.2

Projects

None yet



PERCONA
LIVE

Terraform AWS Aurora MySQL Modules



<https://github.com/sanjeet-deshpande/tf-aws-aurora>

Restore Using Percona XtraBackup



Advantages:

- Physical migration is faster than logical migration
- No impact on Database performance when a backup is taken for physical migration.
- Physical migration including complex database components

Limitations:

- Source and destination databases must be MySQL 5.6.
- You can't restore into an existing Aurora instance using this method.
- User accounts, functions, and stored procedures are not imported automatically.
- Partial migration of Source Database is not supported

Final Thoughts

Contributions to this talk:



Alkin Tezuysal (Percona)

Alkin has extensive experience in enterprise relational databases working in various sectors for large corporations. With more than 20 years of industry experience he has acquired skills for managing large projects from ground up to production. For the past eight years he's been focusing on e-commerce, SaaS and MySQL technologies. He managed and architected database topologies for high volume site at eBay Intl. He has several years of experience on 24X7 support and operational tasks as well as improving database systems for major companies. He has led MySQL global operations team on Tier 1/2/3 support for MySQL customers. In July 2016 he has joined Percona's expert technical management team. Contact him at @ask_dba



Sanjeet Deshpande (Autodesk)

Sanjeet is a Senior DevOps having 10+ years of experience and currently working with Autodesk, Singapore. experienced in architecting, deploying and managing the cloud infrastructures/applications and automation experience. he has worked extensively on AWS services like Lambda, SQS, RDS, SNS to name a few. Worked closely with engineering team of different application to suggest changes to improve uptime of the application.



PERCONA
LIVE EUROPE
FRANKFURT

November 5-7th, 2018

Call for Papers Open!

Connect. Accelerate. Innovate.

- MySQL, MongoDB, PostgreSQL & other Open Source Databases
- Security, Open Source Databases, Serverless, Cloud or On Premise
- High Availability, Scalability
- Business Goals, Case Studies, What the Future Holds
- Radisson Blu Hotel, Frankfurt, Germany

Submit Your Proposal by August 10th!

Get Your Tickets for Percona Live Europe!

- We're delighted to be in Frankfurt this year! A vibrant city, in a central location with many direct flights makes it easy for you to get here!
- Percona Live Europe 2018 includes a new business track that covers the best ideas for how open source databases and database technologies can address and solve business issues such as application time-to-market, resource costs, OPEX and CAPEX expenses, etc.

Super Saver Tickets Available Until
August 19th!

Prices Increase on the 20th!

www.percona.com/live/e18/

Q&A

Thank You!
