Cloud Integration, Migration & Upgrades

End-to-End Database Migration to Managed Cloud Platforms

Why Cloud Database Migration

Organizations migrate databases to the cloud to improve availability, scalability, security, and operational efficiency. Managed services reduce administrative overhead while enabling faster innovation and resilience compared to self-managed infrastructure.

Scope of This Presentation

This deck covers cloud database options, structured migration planning, step-by-step AWS RDS and Aurora migrations, zero-downtime approaches using AWS DMS, and safe in-place upgrades from MySQL 5.7 to 8.0.

Platforms Covered

Primary focus is on AWS managed database services, including RDS MySQL, Aurora MySQL, and Database Migration Service, with references to Azure Database for MySQL and Google Cloud SQL for comparison.

Target Audience

Designed for cloud engineers, database administrators, and architects responsible for migrating, upgrading, and operating production MySQL workloads in cloud environments.

Cloud Database Options

Managed MySQL Services Across Major Cloud Providers

- **AWS RDS MySQL:** Amazon RDS MySQL is a fully managed relational database service that handles backups, patching, monitoring, and high availability. It is well-suited for lift-and-shift migrations and traditional MySQL workloads requiring minimal architectural change.
- **Amazon Aurora MySQL:** Aurora MySQL provides MySQL compatibility with significantly higher performance and availability. Data is replicated six times across three availability zones, enabling fast failover, read scaling, and lower operational risk for large workloads.
- **Azure Database for MySQL:** Azure offers managed MySQL services including Flexible Server and Hyperscale options. These are optimized for Azure-native applications and integrate tightly with Azure networking, identity, and monitoring services.
- **Google Cloud SQL for MySQL:** Cloud SQL is Google’s fully managed MySQL service, providing automated backups, replication, and patching. It is ideal for teams operating within the Google Cloud ecosystem seeking simplicity and reliability.

Pre-Migration Checklist

Critical Planning Steps Before Moving to the Cloud

- **Assess Current Database:** Evaluate database size, growth rate, transaction volume, connection count, and latency requirements. Identify peak usage patterns and performance bottlenecks to ensure the target cloud configuration can meet or exceed existing SLAs.
- **Plan Target Configuration:** Define CPU, memory, storage type, and IOPS requirements. Configure backup retention, monitoring, and alerting strategies early to align with production-grade operational and compliance needs.
- **Downtime & Rollback Planning:** Determine acceptable downtime windows and define a clear rollback strategy. Establish a communication plan for stakeholders to minimize business disruption during migration activities.
- **Test in Staging Environment:** Perform trial migrations using a subset of production data. Validate application behavior, query performance, and data integrity before executing the final production cutover.

AWS RDS Migration – Source Preparation

Preparing the Existing MySQL Database for Cloud Migration

- **Create Consistent Logical Backup:** Use mysqldump with --single-transaction to create a consistent backup without locking tables. This approach is safe for InnoDB and minimizes disruption to production workloads during the backup process.
- **Verify Backup Integrity:** Validate the backup file size and test restoring it into a non-production MySQL instance. This ensures the dump is usable and prevents late discovery of corrupted or incomplete backups.
- **Enable Binary Logs for PITR:** Preserve MySQL binary logs to support point-in-time recovery. Binary logs are critical for restoring incremental changes and enabling near-zero-downtime migration strategies.
- **Baseline Performance Metrics:** Capture baseline metrics such as query latency, throughput, and resource utilization. These metrics provide a comparison point to validate performance after migrating to AWS RDS.

AWS RDS Migration – Create & Restore

Provisioning the RDS Instance and Restoring Data

- **Provision RDS MySQL Instance:** Create a new RDS MySQL instance using the appropriate engine version, instance class, and storage size. Enable Multi-AZ deployment, encryption at rest, automated backups, and define maintenance and backup windows aligned with off-peak hours.
- **Network & Security Configuration:** Place the RDS instance in private subnets, configure security groups to restrict access, and integrate with IAM and KMS. Proper networking and security controls are critical for production-grade deployments.
- **Restore Backup to RDS:** Load the mysqldump backup into the RDS endpoint using the MySQL client. Monitor restore progress and validate schema and row counts to confirm data integrity.
- **Validate Application Connectivity:** Update application connection strings to point to the RDS endpoint and perform functional testing. Confirm query performance and error logs before proceeding to final cutover.

Aurora MySQL vs RDS MySQL

Choosing the Right Managed MySQL Engine on AWS

- **Performance Characteristics:** Aurora MySQL delivers up to 3–5x better read performance compared to standard RDS MySQL by using a distributed, log-structured storage layer optimized for cloud-scale workloads.
- **Availability & Durability:** Aurora automatically replicates data six times across three availability zones, enabling faster failover and higher durability than traditional single-instance RDS deployments.
- **Scalability Model:** RDS MySQL scales vertically by changing instance sizes, while Aurora supports rapid read scaling through reader instances and serverless options for variable workloads.
- **Cost & Use Case Fit:** RDS MySQL is cost-effective and simple for small to medium workloads. Aurora becomes more cost-efficient at scale, particularly for read-heavy or mission-critical applications.

Migrating to Amazon Aurora

High-Availability MySQL Migration with Aurora Clusters

Create Aurora MySQL Cluster

Provision an Aurora MySQL cluster with the desired engine version and instance class. Aurora requires a cluster-based architecture with separate writer and reader endpoints, providing built-in high availability.

Restore Data into Aurora

Restore existing MySQL data using the same logical backup process as RDS. Aurora maintains compatibility with standard MySQL clients and tools, simplifying migration workflows.

Reader Endpoints & Scaling

Aurora automatically manages read replicas and exposes a reader endpoint for load-balanced read traffic. Additional reader instances can be added to scale read-heavy workloads with minimal effort.

Cross-Region Replication

Aurora supports cross-region read replicas for disaster recovery and global applications. This enables low-latency regional reads and fast recovery from regional failures.

AWS Database Migration Service (DMS)

Near Zero-Downtime MySQL Migration with CDC

- **Why Use AWS DMS:** AWS DMS enables near zero-downtime migrations by combining an initial full data load with continuous change data capture. This approach minimizes business disruption for production systems.
- **Source & Target Endpoints:** Configure secure source and target endpoints for on-premises MySQL and AWS RDS or Aurora. Endpoint validation ensures connectivity, credentials, and network configuration are correct before migration begins.
- **Migration Tasks & CDC:** Create migration tasks using Full Load + CDC to replicate ongoing changes. DMS continuously applies inserts, updates, and deletes to the target database while applications remain online.
- **Monitoring & Cutover:** Monitor full load progress, CDC lag, and row count validation. Once replication lag approaches zero, application writes can be stopped briefly to perform final cutover safely.

Upgrading MySQL 5.7 to 8.0

Safe In-Place Upgrade Strategy and Risk Mitigation

- **Pre-Upgrade Compatibility Checks:** Review deprecated features, reserved keywords, and authentication changes introduced in MySQL 8.0. Run mysqlcheck and application test suites to identify breaking changes early.
- **Comprehensive Backup Strategy:** Perform full logical backups, preserve binary logs, and back up configuration files. Reliable backups are mandatory to support fast rollback in case of upgrade failure.
- **Controlled Upgrade Execution:** Stop MySQL services, install MySQL 8.0 packages, update configuration parameters, and run mysql_upgrade. Avoid starting the database until upgrade scripts complete successfully.
- **Post-Upgrade Validation:** Verify MySQL version, monitor logs for warnings, test application functionality, and evaluate performance changes such as new execution plans or optimizer behavior.

Post-Migration Optimization & Security

Stabilizing, Tuning, and Securing Cloud MySQL Workloads

Performance Monitoring
Enable CloudWatch metrics, enhanced monitoring, and performance insights to track query latency, CPU utilization, memory pressure, and I/O behavior. Continuous visibility is essential after migration.

Query & Index Optimization
Analyze slow query logs and execution plans to identify inefficient queries. Adjust indexes, buffer pool sizing, and caching strategies to align with cloud workload patterns.

Security Hardening
Restrict network access using security groups, enforce strong authentication methods, rotate credentials, and enable encryption in transit and at rest to meet security best practices.

Operational Best Practices
Automate backups, patching, and alerts. Regularly test restores and failover scenarios to ensure the database remains resilient and recoverable under failure conditions.

Key Takeaways & Best Practices

Executing Reliable Cloud Database Migrations

- **Choose the Right Target Platform:** Select RDS MySQL for simplicity and cost efficiency, or Aurora MySQL for high availability, read scalability, and mission-critical workloads. Platform choice should align with workload scale and business risk tolerance.
- **Plan Before You Migrate:** Successful migrations depend on upfront assessment, capacity planning, downtime analysis, and rollback strategies. Skipping planning is the most common cause of migration failure.
- **Minimize Downtime with DMS:** Use AWS Database Migration Service with CDC for near zero-downtime migrations. Continuous replication allows safe cutover while maintaining data consistency.
- **Validate, Monitor, and Optimize:** Always validate data integrity, monitor performance post-migration, and continuously optimize queries, indexes, and configurations to fully realize cloud benefits.