Cloud Integration, Migration & Upgrades

10.1 Migration to Cloud Databases

Cloud Database Options

AWS:

- **RDS MySQL**: Managed relational database
- **Aurora MySQL**: High-performance, auto-scaling MySQL
- **DMS**: Database Migration Service

Azure:

- **Azure Database for MySQL**: Managed service
- **MySQL Hyperscale**: For large-scale applications

Google Cloud:

- **Cloud SQL for MySQL**: Fully managed
- **AlloyDB**: PostgreSQL-compatible (alternative)

Pre-Migration Checklist

1. **Assess Current Database**
    - Size and growth rate
    - Performance requirements
    - Connection count and latency
2. **Plan Target Configuration**
    - CPU, memory, storage
    - Backup retention
    - Monitoring and alerting
3. **Prepare for Downtime**
    - Maintenance window
    - Rollback procedure
    - Communication plan

4. **Test in Staging**
    - Migrate subset of data
    - Test applications
    - Measure performance

10.2 Migrating to AWS RDS

Step 1: Prepare Source Database

```
# Create backup without locks (safe for InnoDB)
mysqldump -u root -p \
  --single-transaction \
  --default-character-set=utf8mb4 \
  --all-databases > migration_backup.sql

# Verify backup
ls -lh migration_backup.sql

# Create binary log backup for PITR
cp /var/log/mysql/mysql-bin.* /backup/
```

Step 2: Create RDS Instance

In AWS Console:

1. Select RDS service
2. Create database → MySQL
3. Choose configuration:
    - DB engine version: MySQL 8.0
    - Instance class: db.t3.micro (for testing) or db.t3.medium (production)
    - Allocated storage: Appropriate size
    - Backup retention: 7+ days

- Multi-AZ: Yes (for production)
- Encryption: Yes

4. Database options:
    - Initial database name: (optional)
    - Parameter group: Create custom if needed
    - Backup window: Off-peak hours
    - Maintenance window: Off-peak hours

Step 3: Restore Backup to RDS

 # Get RDS endpoint (e.g., mydb.abc123.us-east-1.rds.amazonaws.com)

 # Restore backup

```
mysql -h mydb.abc123.us-east-1.rds.amazonaws.com \
   -u admin -p < migration_backup.sql
```

 # Verify data

```
mysql -h mydb.abc123.us-east-1.rds.amazonaws.com \
   -u admin -p -e "SELECT COUNT(*) FROM information_schema.tables;"
```

Step 4: Update Application Connection String

Old: localhost:3306

New: mydb.abc123.us-east-1.rds.amazonaws.com:3306

```
# Test connection
mysql -h mydb.abc123.us-east-1.rds.amazonaws.com \
   -u admin -p database_name -e "SELECT 1;"
```

Step 5: Create Read Replicas (Optional)

In AWS RDS console:

1. Select DB instance
2. Instance Actions → Create Read Replica
3. Specify different AZ or region
4. Create

Step 6: Enable Enhanced Monitoring

In AWS Console:

1. Modify DB instance
2. Enable Enhanced monitoring
3. Enable Cloudwatch logs (error, slow query, general)
4. Enable Performance Insights

10.3 Migrating to AWS Aurora

Aurora vs RDS MySQL

Aurora Advantages:

- 3-5x faster read performance
- Better availability (6 copies across 3 AZs)
- Automatic scaling (Aurora Serverless)
- Compatible with MySQL
- Lower cost for large workloads

Migration Steps

Step 1: Create Aurora Cluster

In AWS Console:

1. RDS → Create database
2. Engine: Aurora MySQL
3. Version: Aurora MySQL 8.0

4. Multi-AZ Deployment: Yes
5. Instance class: db.t3.medium (minimum for Aurora)

Step 2: Restore Backup to Aurora

```
# Get Aurora endpoint
# Restore backup (same as RDS)
mysql -h myaurora.abc123.us-east-1.rds.amazonaws.com \
  -u admin -p < migration_backup.sql

# Verify
mysql -h myaurora.abc123.us-east-1.rds.amazonaws.com \
  -u admin -p -e "SELECT @@version;"
```

Step 3: Configure Read Replicas

Aurora automatically maintains read replicas. Create additional reader endpoint:

In AWS Console:

1. Cluster details
2. Add instance (automatically added to read replica)

Step 4: Setup Cross-Region Replication (Optional)

```
# Create cross-region read replica
# Aurora automatically replicates to standby region
# (Configured in AWS Console)
```

10.4 Using AWS Database Migration Service (DMS)

DMS Advantages

- Zero-downtime migration
- Supports heterogeneous databases (MySQL → Oracle, etc.)
- Ongoing replication capability
- Schema conversion tools

DMS Migration Process

Step 1: Create Source Endpoint

1. DMS → Endpoints → Create endpoint
2. Source engine: MySQL
3. Server name: On-premises IP/hostname
4. Port: 3306
5. Username: root
6. Password: credentials
7. Test connection

Step 2: Create Target Endpoint

1. DMS → Endpoints → Create endpoint
2. Target engine: Aurora MySQL
3. Server name: RDS endpoint
4. Provide credentials
5. Test connection

Step 3: Create Migration Task

1. DMS → Migration tasks → Create task
2. Source: On-premises endpoint
3. Target: RDS/Aurora endpoint
4. Migration type: Full load + change data capture (CDC)
5. Select schemas/tables to migrate

Step 4: Monitor Migration

- Full load progress
- CDC status
- Data validation
- ROW count comparison

Step 5: Complete Migration

1. Stop application writes to source

2. Monitor CDC lag (should be near zero)
3. Perform final validation
4. Update application connection string
5. Switch to target database

10.5 Upgrading MySQL 5.7 to 8.0

Pre-Upgrade Planning

1. **Compatibility Check**
    - Review deprecated features
    - Check for reserved keywords
    - Test application queries
2. **Backup**
    - Full backup of all databases
    - Binary logs
    - Configuration files
3. **Test in Staging**
    - Test upgrade procedure
    - Run application tests
    - Measure performance

Upgrade Process

Step 1: Prepare for Upgrade

```
# Check current version
mysql -e "SELECT VERSION();"
# MySQL 5.7.35


# Check for compatibility issues
mysqlcheck -u root -p --all-databases
```

Step 2: Backup

 # Full backup

mysqldump -u root -p --all-databases > backup_57.sql

 # Backup configuration

sudo cp /etc/mysql/my.cnf /etc/mysql/my.cnf.backup

 # Backup binary logs

sudo cp /var/log/mysql/mysql-bin.* /backup/

Step 3: Stop MySQL

sudo systemctl stop mysql

Step 4: Update Configuration

sudo vim /etc/mysql/my.cnf

Add MySQL 8.0 specific settings:

[mysqld]

 # MySQL 8.0 default plugin

default_authentication_plugin = mysql_native_password

 # SQL Modes

sql_mode =
'STRICT_TRANS_TABLES,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION'

 # Other settings

character_set_server = utf8mb4

collation_server = utf8mb4_unicode_ci

Step 5: Install MySQL 8.0 Packages

sudo apt install -y mysql-server=8.0.*

 # Do NOT start yet

Step 6: Run Upgrade Script

 # Perform in-place upgrade

sudo mysql_upgrade -u root -p

 # Output shows:

 # Checking databases

 # Upgrading the sys schema

 # Checking mysql schema

 # Checking performance_schema schema

 # etc.

Step 7: Start MySQL 8.0

sudo systemctl start mysql

 # Verify version

mysql -u root -p -e "SELECT VERSION();"

 # Should show MySQL 8.0

Step 8: Test Applications

 # Run application tests

 # Check logs for deprecation warnings

tail -100 /var/log/mysql/error.log

 # Monitor performance

SHOW ENGINE INNODB STATUS\G

Rollback Procedure (if needed)

 # Stop MySQL 8.0

sudo systemctl stop mysql

 # Reinstall MySQL 5.7

sudo apt install -y mysql-server=5.7.*

 # Restore data

sudo rm -rf /var/lib/mysql

mysql < backup_57.sql

 # Restore configuration

sudo cp /etc/mysql/my.cnf.backup /etc/mysql/my.cnf

 # Start MySQL 5.7

sudo systemctl start mysql

Post-Upgrade Tasks

1. **Performance Tuning**
    - Analyze new execution plans
    - Adjust indexes if needed
    - Monitor buffer pool hit ratio
2. **Feature Migration**
    - Migrate to new authentication plugin (caching_sha2_password)
    - Use window functions and CTEs

- Implement instant DDL

3. **Security**
    - Update user authentication
    - Review privilege grants
    - Enable audit logging

10.6 Summary: Key Takeaways

1. **Cloud Options**: RDS MySQL, Aurora, Azure MySQL, Google Cloud SQL
2. **RDS Migration**: Backup → Create RDS → Restore → Update application
3. **Aurora**: Better performance, automatic HA, auto-scaling
4. **DMS**: Zero-downtime migration with CDC capability
5. **Upgrade Process**: Backup → Stop → Install → Run mysql_upgrade → Start → Test
6. **Rollback Plan**: Keep backup and old packages for quick rollback
7. **Post-Migration**: Monitor performance, update applications, optimize configurations

APPENDIX: Reference Materials

A. Common MySQL Commands Reference

Connection Commands

mysql -u username -p -h hostname -P 3306 -D database

mysql --defaults-file=~/.my.cnf

mysql -e "SQL_QUERY" database_name

Database Commands

SHOW DATABASES;

CREATE DATABASE db_name;

DROP DATABASE db_name;

USE database_name;

ALTER DATABASE db_name CHARACTER SET utf8mb4;

Table Commands

SHOW TABLES;

DESCRIBE table_name;

SHOW CREATE TABLE table_name;

CREATE TABLE ...

ALTER TABLE table_name ...

DROP TABLE table_name;

TRUNCATE TABLE table_name;

User Commands

CREATE USER 'user'@'host' IDENTIFIED BY 'password';

ALTER USER 'user'@'host' IDENTIFIED BY 'new_password';

DROP USER 'user'@'host';

GRANT privileges ON database.table TO 'user'@'host';

REVOKE privileges ON database.table FROM 'user'@'host';

SHOW GRANTS FOR 'user'@'host';

B. Performance Tuning Parameters

[mysqld]

 # Memory

innodb_buffer_pool_size = 8G *# 60-80% of RAM*

sort_buffer_size = 2M *# Per-connection*

tmp_table_size = 32M *# Temporary tables*

max_heap_table_size = 32M *# Memory table limit*

 # Connections

max_connections = 200 *# Total connections*

max_user_connections = 50 *# Per-user limit*

max_allowed_packet = 256M *# Max query size*

 # Logging

slow_query_log = 1 *# Enable slow log*

long_query_time = 2 *# Log queries > 2 sec*

log_bin = /var/log/mysql/mysql-bin *# Binary log*

 # InnoDB

innodb_log_file_size = 256M *# Redo log size*

innodb_flush_log_at_trx_commit = 2 *# Balance performance/durability*

innodb_flush_method = O_DIRECT *# Bypass OS cache*

C. Security Checklist

- [ ] Remove anonymous users
- [ ] Remove test database
- [ ] Disable remote root login
- [ ] Use strong passwords (validate_password plugin)
- [ ] Restrict user host access
- [ ] Use SSL for remote connections
- [ ] Enable audit logging
- [ ] Regular password rotation
- [ ] Principle of least privilege
- [ ] Monitor user activity

D. Troubleshooting Commands

 # Check MySQL status

sudo systemctl status mysql

sudo service mysql status

 # Check logs

tail -50 /var/log/mysql/error.log

tail -f /var/log/mysql/error.log

 # Check processes

ps aux | grep mysql

ps aux | grep mysqld

 # Check ports

sudo netstat -tulnp | grep 3306

sudo ss -tulnp | grep 3306

 # Check disk usage

du -sh /var/lib/mysql/*

df -h /var/lib/mysql

 # Restart MySQL

sudo systemctl restart mysql

sudo service mysql restart