love your data

Lessons learned when managing MySQL in the Cloud

March 2025

Pythian

love your data

About Me

👨‍💻 **Principal Consultant: Pythian (OSDB Practice)**

🎓 **Education: MSc in Software Engineering**

📜 **Certifications**

- **MongoDB Certified DBA**
- **Oracle Professional MySQL 5.7**
- **Terraform Associate Certified**
- **GCP Professional Architect**

💻 **Expertise: Bash, Python, Hybrid Cloud**

🌟 **Personal: Husband and Father, Avid Traveler, Speaker**

🌐 **Social**

@igorle doncovski in/igorle

About

🛠️ **Hands-On Experience**

💡 **No GenAI Content (except…)**

☁️ **Cloud-Focused (DBaaS)**

❌ **Not a Training Session**

🌱 **Beginner Friendly**

Agenda

- 🎤 Introduction
- ⚙️ Configuration and Installation
- 🤖 Automation
- 📦 Version Control and Upgrades
- 📊 Performance Tuning
- 📈 Cost-Effective Scalability
- 🌍 High Availability
- 📉 Monitoring
- ♻️ Disaster Recovery
- 🔒 Security
- 💬 Q&A

Benefits of Cloud Adoption

Automation

Scalability

Cost Efficiency

Collaboration

Innovation

Security and Compliance

Resilience and DR

Infrastructure as Code

- Terraform (OpenTofu)
- Cloud Formation
- Cloud Development Kit
- Helm (Kubernetes)
- Ansible
- Chef
- Puppet
- Reverse Engineering

Infrastructure as Code

```
provider "aws" {
  region = "us-west-1"  # Change to your preferred region
}

resource "aws_instance" "example" {
  ami           = "ami-0c55b159cbfafe1f0"  # Valid AMI ID for the region
  instance_type = "t2.micro"

  root_block_device {
    volume_size = 20  # OS disk size in GB
  }

  ebs_block_device {
    device_name = "/dev/sdb"
    volume_size = 500  # MySQL partition size in GB
  }

  network_interface {
    network_interface_id = aws_network_interface.example.id
    device_index         = 0
  }

  tags = {
    Name = "MyEC2Instance"
  }
}

resource "aws_network_interface" "example" {
  subnet_id   = "subnet-d744039d"  # Replace with your subnet ID
  private_ips = ["10.0.1.100"]
}
```

```
provider "google" {
  project = "my-gcp-project"  # Replace with valid GCP project ID
  region  = "us-west1"
}
resource "google_compute_instance" "example" {
  name         = "my-gcp-instance"
  machine_type = "e2-medium"
  zone         = "us-west1-a"

  boot_disk {
    initialize_params {
      image = "debian-cloud/debian-11"  # Replace with a valid image
      size  = 20  # OS disk size in GB
    }
  }
  attached_disk {
    source      = google_compute_disk.mysql_disk.id
    device_name = "mysql-disk"
  }
  network_interface {
    network    = "default"
    subnetwork = google_compute_subnetwork.example.id
    network_ip = "10.0.1.100"
  }
}
resource "google_compute_disk" "mysql_disk" {
  name  = "mysql-disk"
  type  = "pd-ssd"
  size  = 500  # MySQL partition size in GB
  zone  = "us-west1-a"
}
resource "google_compute_subnetwork" "example" {
  name          = "example-subnet"
  ip_cidr_range = "10.0.1.0/24"
  network       = "default"
  region        = "us-west1"
}
```

Self managed MySQL

🛠️ Flexibility & Customization

- Full control over MySQL version, storage, and configurations
- Ability to fine-tune performance settings (OS and Database layer)
- Choose instance types based on workload needs
- Similar experience for all setups, making migration easier

📉 Management Overhead

- OS patching, MySQL upgrades, backups, and monitoring
- Scaling requires downtime or complex automation
- Increased complexity and lack of expertise
- No managed failover, needs custom HA solutions (e.g., Orchestrator)

On-premises

- App optimization
- Scaling
- High availability
- Database backups
- DB software patching
- DB software install
- OS patching
- OS install
- Server maintenance
- Hardware lifecycle
- Power/network/HVAC

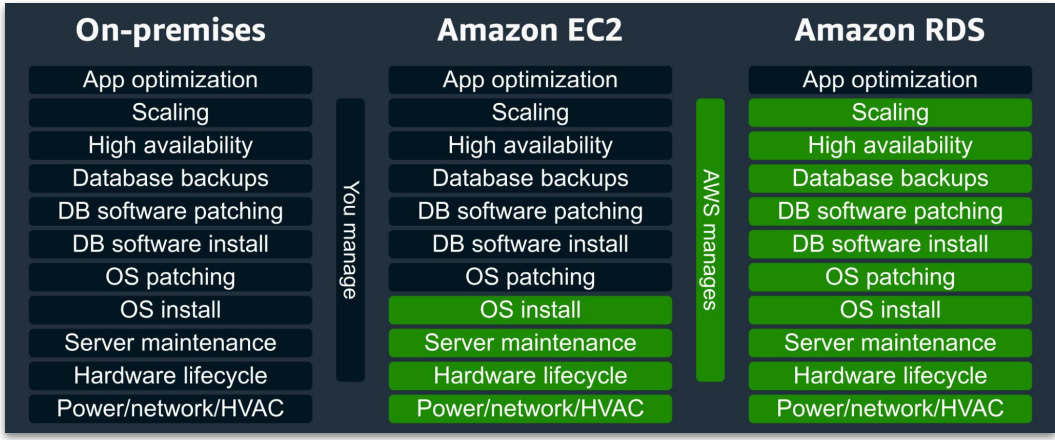
Self Managed vs DBaaS

✅ Self Managed

- Workloads needing custom MySQL tuning
- Applications requiring specific MySQL versions or extensions
- Teams with DBA expertise to handle operations
- Custom OS patching and tuning

🛠️ Managed Database Service

- Automated backups with PITR, failover, and scaling
- Your workload is highly dynamic and benefits from serverless options
- Fully managed security & compliance without manual effort

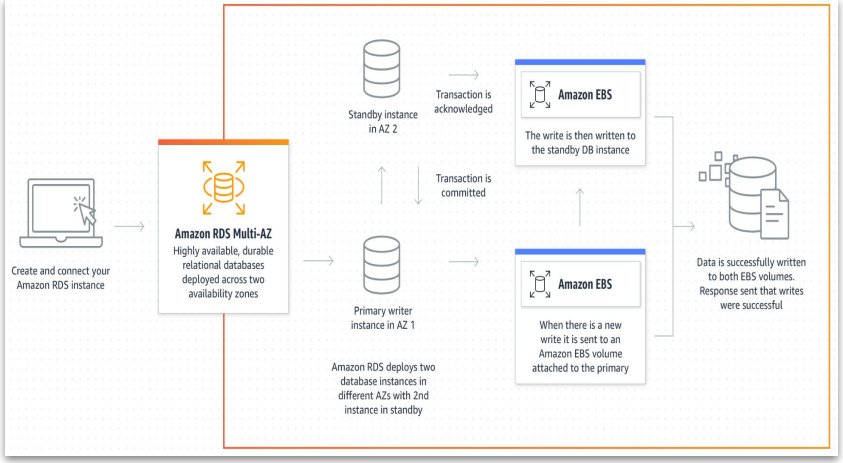


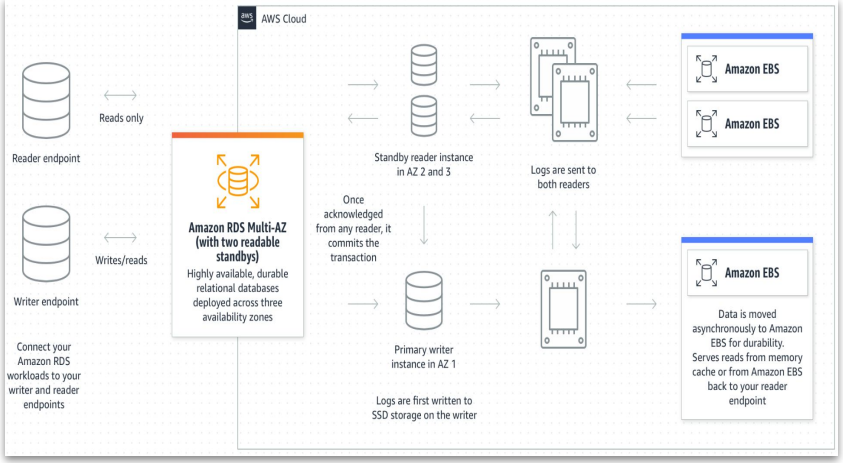
DBaaS Provisioning

- High Availability (Production, Dev/Test, Free Tier)
- Multi AZ deployment (standby, replicas)
- Cost and Pricing (On Demand, Reserved)
- Storage Autoscaling
- Parameter Group (Flags) for GLOBAL VARIABLES
- Primary instance, Secondary instances
- Data Migration (DMS, Logical Backup, xtrabackup ...)
- Performance (Query) Insights (7 days free tier)
- Backup storage (up to 35 days with no extra cost)

High Availability (AWS RDS)

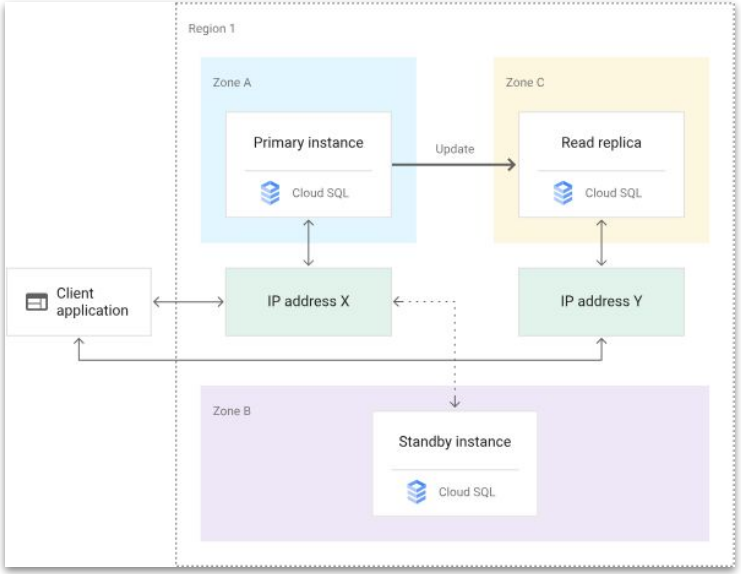
- Multi AZ deployment (single standby or two readable standbys)
- Read replicas - asynchronous replication





High Availability (GCP CloudSQL)

- Multi AZ deployment - synchronous replication to a standby node
- Read replicas - asynchronous replication



Pricing

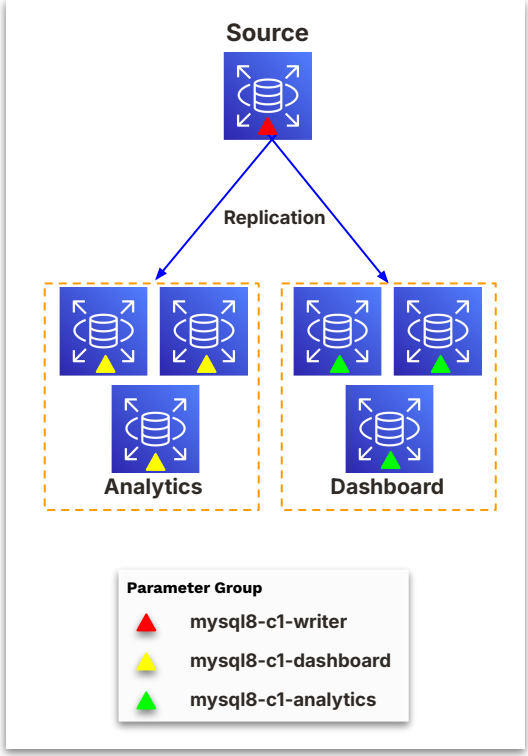
- **Storage and IO is not included in the Price (USD)**

Resource	On-Demand Price	1-Year Commitment	1-Year Discount (%)	3-Year Commitment	3-Year Discount (%)
vCPUs	$36.208 per vCPU	$27.16	25%	$17.38	52%
Memory	$6.132 per GB	$4.60	25%	$2.94	52%
HA vCPUs	$72.343 per vCPU	$54.26	25%	$34.72	52%
HA Memory	$12.264 per GB	$9.20	25%	$5.89	52%

Resource	On-Demand Hourly Rate	1-Year Reserved Instance	1-Year Savings (%)	3-Year Reserved Instance	3-Year Savings (%)
db.t3.micro	$0.02	$0.01	29%	$0.01	34%
db.m5.large	$0.19	$0.13	30%	$0.12	37%
db.r5.xlarge	$0.48	$0.33	30%	$0.30	37%

Parameter Groups

- Default parameter group - non modifiable
- Custom parameter group (some options also non modifiable)
- Best practice, one parameter group per instance (or group of instances)
- At least one parameter group per Writer and Reader
- Some changes are applied only after instance reboot
- Cluster parameter groups
- Compare parameter groups



Database migration

- **Cloud native DMS tools**
  - Does not always work, hard to troubleshoot (Oracle, MSSQL → MySQL)
- **Xtrabackup**
  - Works with AWS, also hard to troubleshoot
- **mydumper**
  - Logical multi threaded, issues with JSON column type
- **Mysqldump**
  - Single threaded, takes long for large datasets
- **MySQL Workbench**
  - Single threaded, graphical interface for database migration. (Oracle, MSSQL → MySQL)
- **MySQL Shell**
  - Multi threaded, logical backup

Database migration plan

- **Create custom parameter group upfront**
  - log_bin_trust_function_creators=ON (if importing stored procedures)
  - SUPER, SYSTEM_VARIABLES_ADMIN, FILE privileges are restricted
- **Choose the proper instance size that matches your on-prem instance (CPU, Memory, Disk, Network)**
- **Skip the multi-AZ initially if loading huge data**
- **Create an instance with the same DB minor version as your source instance**
- **Backups and Performance Insights are free with 7 days retention period**
- **Maintenance - disable auto minor version upgrades**
- **Use Deletion protection to prevent accidental deletion of the instance**

Database migration execution

- Mysql Shell (Migrating from Google CloudSQL to AWS RDS)
- Single command, util.copyInstance()

```
mysqlsh --uri mysql://<username>:<password>@<dbass-endpoint-gcp>:3306

util.copyInstance('mysql://<username>@<dbass-endpoint-rds>:3306', {checksum: true, compatibility:
["strip_definers"], users: false, dropExistingObjects: true, dryRun : true})

pt-show-grants -h<dbass-endpoint-gcp> -u<username> --ask-pass > users.sql

mysql -u<username> -h<dbaas-endpoint-rds> -p < users.sql
```

Replication

MySQL < 8.4

```
CHANGE MASTER to MASTER_HOST=<external_host>, MASTER_USERNAME=<username>, MASTER_PASSWORD=<password>,

MASTER_PORT=<port>, MASTER_LOG_FILE=<binlog>, MASTER_LOG_POS=<position>;

START SLAVE;
```

MySQL >= 8.4

```
CHANGE REPLICATION SOURCE to SOURCE_HOST=<external_host>, SOURCE_USERNAME=<username>,

SOURCE_PASSWORD=<password>, SOURCE_PORT=<port>, SOURCE_LOG_FILE=<binlog>, SOURCE_LOG_POS=<position>;

START REPLICA;
```

Replication

MySQL < 8.4

```
CHANGE MASTER to MASTER_HOST=<exter[illegible]t>, MASTER_USERNAME=<[illegible]e>, MASTER_PASSWORD=<password>,
MASTER_PORT=<port>, MASTER_LOG_FILE=<bi[illegible]MASTER_LOG_PO[illegible]tion>;
START SLAVE;
```

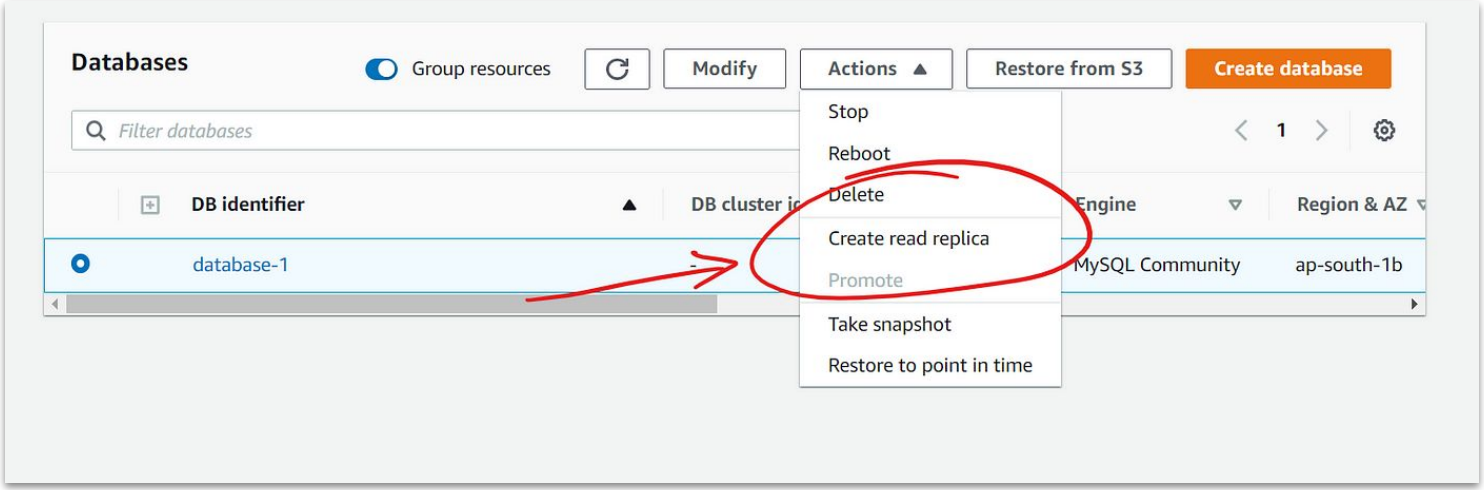
MySQL >= 8.4

```
CHANGE REPLICATION SOURCE to SOURCE_HO[illegible]rnal_host>, SO[illegible]ERNAME=<username>,
SOURCE_PASSWORD=<password>, SOURCE[illegible]port>, SOURCE_LOG_FILE=[illegible]>, SOURCE_LOG_POS=<position>;
START REPLICA;
```

Replication

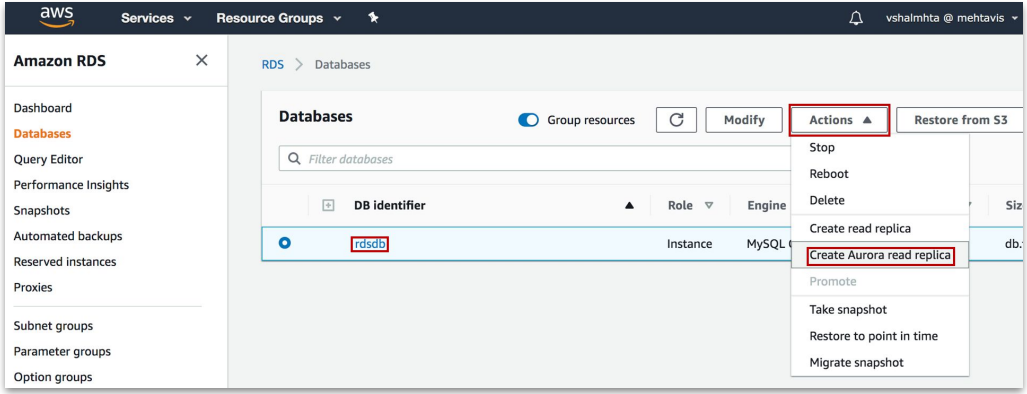
- CALL mysql.rds_set_external_master (*host_name, host_port, replication_user_name, replication_user_password, mysql_binary_log_file_name, mysql_binary_log_file_location, ssl_encryption*);
- CALL mysql.rds_set_external_source_gtid_purged(*server_uuid, start_pos, end_pos*);
- CALL mysql.rds_set_external_master_with_auto_position (*host_name, host_port, replication_user_name, replication_user_password, ssl_encryption, delay*);
- CALL mysql.rds_start_replication;
- CALL mysql.rds_reset_external_master;

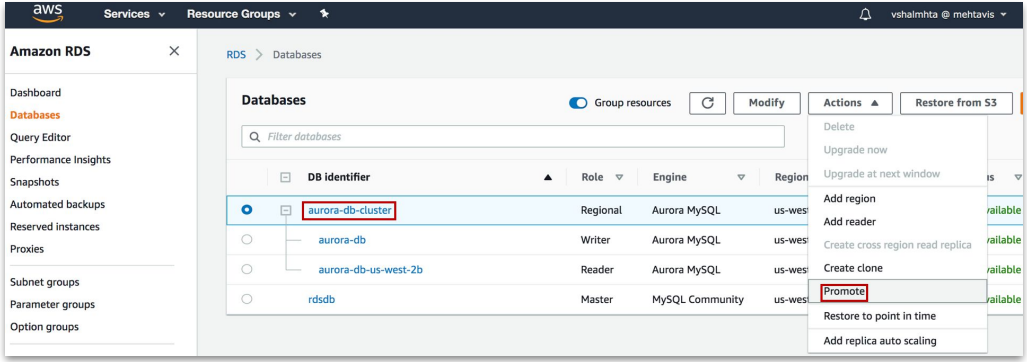
Replication



Replication - switch to Aurora

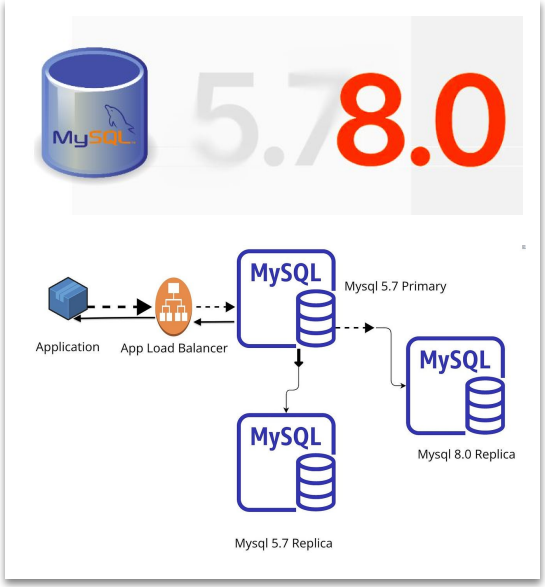
- Create Aurora read replica
- Set RDS Primary as read-only
- Confirm replication is in sync
- Promote Aurora as Standalone
- Point Apps to Aurora





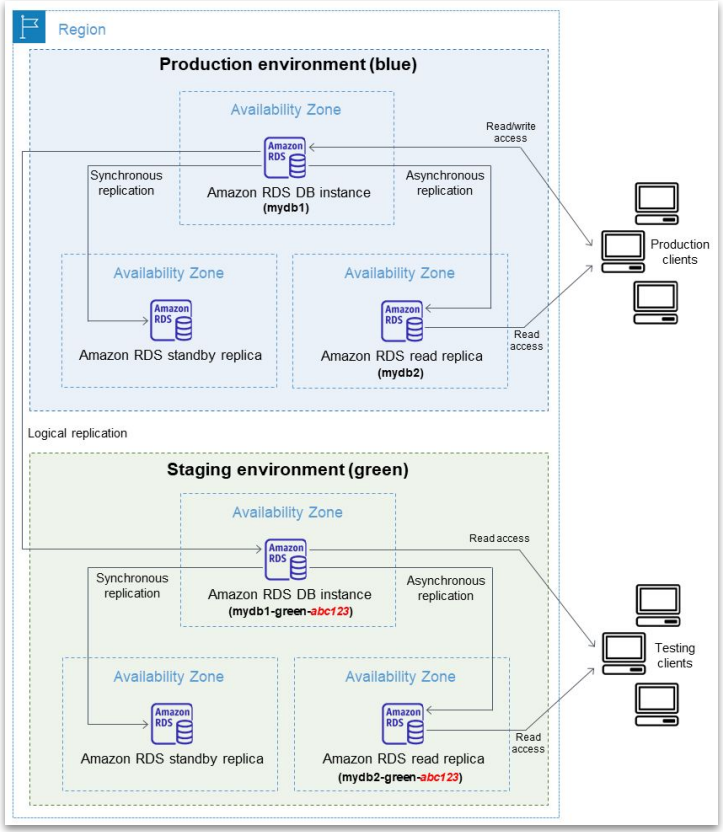
Major version upgrade

- **Latest major version might not be available as soon as the community version**
- **Upgrading to next major version might fail if:**
  - The current version is not the latest minor version
  - The current instance class is not supported (only current and next generation classes are supported)
  - Custom parameter group is not created for the next version
- **Restore a test instance from snapshot**
- **Attempt an upgrade on the test instance**
- **The provider does pre-check for upgrade and report on issues**
- **Verify application compatibility and deprecated features**
- **Upgrade secondary nodes prior upgrading Primary**
- **Promote a Secondary node**
- **Extended support for EOL version ($$)**



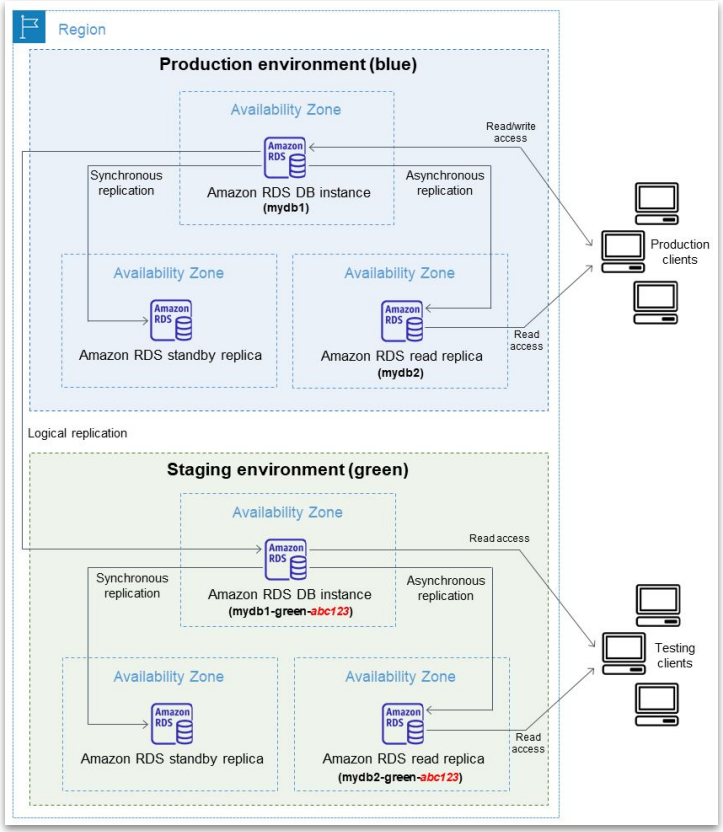
Blue - Green Deployments

- Blue - current environment
- Green - staging environment
- Green environment can be changed without affecting the Blue environment
- Test the Green environment
- Switchover in less than a minute with no App changes
- Can be useful for schema changes on big tables
- Major version upgrades with failback option



BG Deployments - major upgrade

- **Create the blue-green deployment**
  - Verify binary logging is ON
  - Create new parameter groups
  - Create a blue-green deployment
  - Change the binlog retention for the new cluster
- **Switchover to next major MySQL**
  - Switchover to the green environment
  - Capture the binary log file name and position
  - Delete the blue-green deployment
  - Set the previous (old) server to NOT writeable
  - Verify the old server has been set to read-only
- **Setup replication from new MySQL to old**
  - Create a replication user on the new writer
  - Set up reverse replication



Percona toolkit

Collection of advanced open source command-line tools

```
pt-mysql-summary -h<endpoint.amazonaws.com> -u<username> --ask-pass > mysql_summary.txt
```

```
pt-query-digest --processlist  h=<endpoint.amazonaws.com> -u<username> --run-time=300s --daemonize --ask-pass >
query_digest_$(date '+%Y-%m-%d').txt
```

```
pt-duplicate-key-checker -h<endpoint.amazonaws.com> -u<username> --ask-pass > duplicate_indexes.txt
```

```
pt-online-schema-change --alter 'ADD COLUMN id INT' h=<endpoint.amazonaws.com> D=<database>,t=<table>;
```

```
pt-show-grants -h<endpoint.amazonaws.com> -u<username> --ask-pass > users.sql
```

Monitoring - Performance Insights

Dimensions | Metrics

Metrics dashboard

Aurora MySQL database health summary

Feedback | Export to CloudWatch

CPU utilization (Percent)
os.cpuUtilization.steal.avg
os.cpuUtilization.guest.avg
os.cpuUtilization.irq.avg
os.cpuUtilization.wait.avg

Free memory (Bytes)
FreeableMemory

Out-of-memory kills (Count)
os.memory.outOfMemoryKillCount.avg

IO latency (Milliseconds)
os.diskIO.auroraStorage.readLatency.avg
os.diskIO.auroraStorage.writeLatency.avg
os.diskIO.rdstemp.await.avg
CommitLatency

IO operations (Per second)
os.diskIO.auroraStorage.readIOsPS.avg
os.diskIO.auroraStorage.writeIOsPS.avg
os.diskIO.rdstemp.readIOsPS.avg
os.diskIO.rdstemp.writeIOsPS.avg

IO throughput (Bytes per second)
os.diskIO.auroraStorage.readThroughput.avg
os.diskIO.auroraStorage.writeThroughput.avg
os.diskIO.rdstemp.readKbPS.avg
os.diskIO.rdstemp.writeKbPS.avg

IO disk queue depth (Requests)
os.diskIO.auroraStorage.diskQueueDepth.avg
os.diskIO.rdstemp.avgQueueLen.avg

Network throughput (Bytes per second)
os.network.tx.avg
os.network.rx.avg

Connection utilization (Connections)
db.Users.Threads_connected.avg

Sessions (Sessions)
DBLoad
db.Users.Threads_running.avg
db.Users.Aborted_clients.avg

Logins (Connections)
db.Users.Connections.avg
db.Users.Aborted_connects.avg

InnoDB history list length (Length)
db.Transactions.trx_rseg_history_len.avg

Queries (Per second)
db.SQL.Queries.avg

Query latency (Milliseconds)
SelectLatency
InsertLatency
UpdateLatency
DeleteLatency
CommitLatency
DMLLatency
DDLLatency

Query throughput (Per second)
SelectThroughput
InsertThroughput
UpdateThroughput
DeleteThroughput
CommitThroughput
DMLThroughput
DDLThroughput

DML (Rows per second)
db.SQL.Innodb_rows_deleted.avg
db.SQL.Innodb_rows_updated.avg
db.SQL.Innodb_rows_inserted.avg

Active transactions vs. locks (Transactions)
db.Transactions.active_transactions.avg
db.Locks.Innodb_row_lock_waits.avg

Query cache hit ratio (Percent)
db.Cache.query_cache_hit_rate.avg

Buffer pool hit ratio (Percent)
db.Cache.innoDB_buffer_pool_hit_rate.avg

IO cache vs. disk reads (Pages per second)
db.Cache.Innodb_buffer_pool_reads.avg
db.Cache.innoDB_buffer_pool_hits.avg

Row lock time (Milliseconds)
db.Locks.Innodb_row_lock_time.avg

Monitoring - PMM

Common Performance Issues

📊 Workload

- **Slow queries**
  Some queries take a long time to finish, missing index is common reason
- **Complex queries**
  Queries that join many tables or process large amounts of data take longer to run. Partial indexes or queries doing analytics
- **High concurrency**
  When too many users are running queries at the same time, it can slow things down

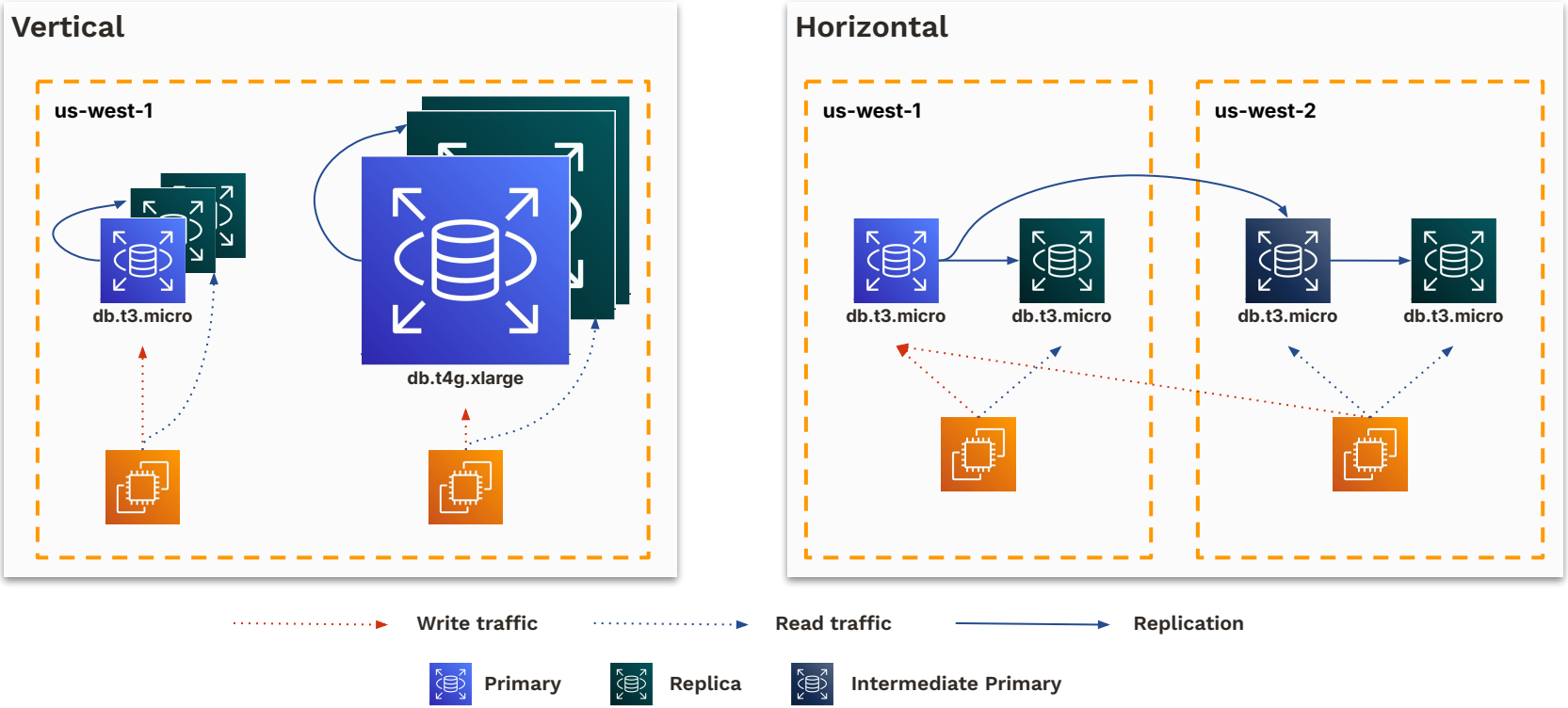
🖥️ System Resources

- **CPU Overload**
  When the database uses too much CPU power, it struggles to process queries efficiently
- **Slow Disk Access**
  If the database has to read or write a lot of data, it can cause delays
- **Not Enough Memory**
  If there isn't enough memory, queries take longer reading data from disk

⚙️ Configuration

- **Wrong Settings**
  Some database settings might not be ideal for the workload, leading to reduced performance
- **Repeated Queries**
  Queries run 1000s of times without adding caching layer

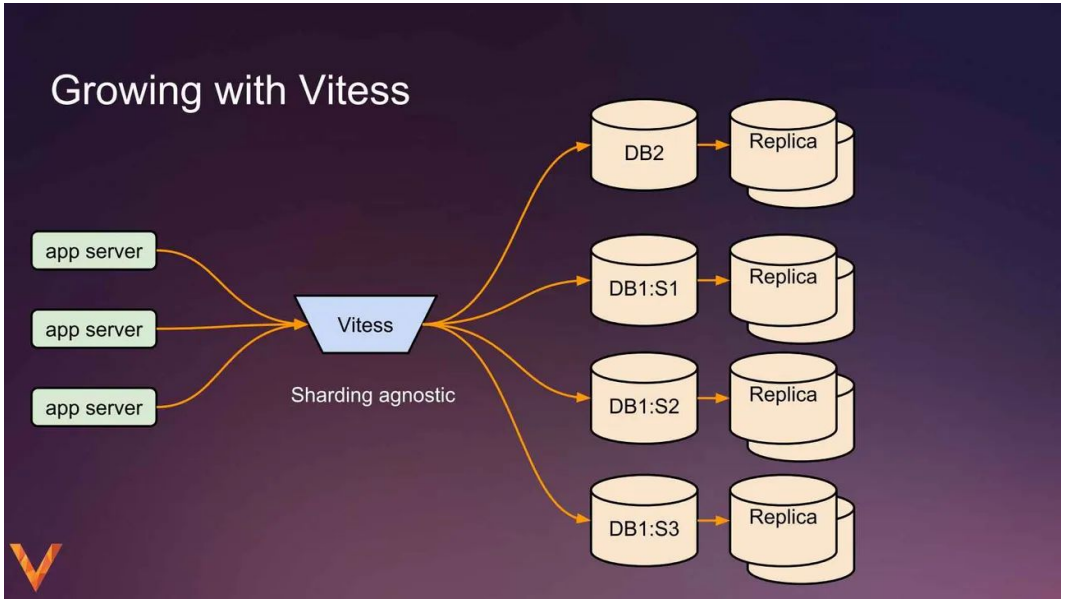
Scaling


Vertical
us-west-1
db.t3.micro
db.t4g.xlarge
Horizontal
us-west-1
us-west-2
db.t3.micro
db.t3.micro
db.t3.micro
db.t3.micro
Write traffic
Read traffic
Replication
Primary
Replica
Intermediate Primary

Vertical Scaling

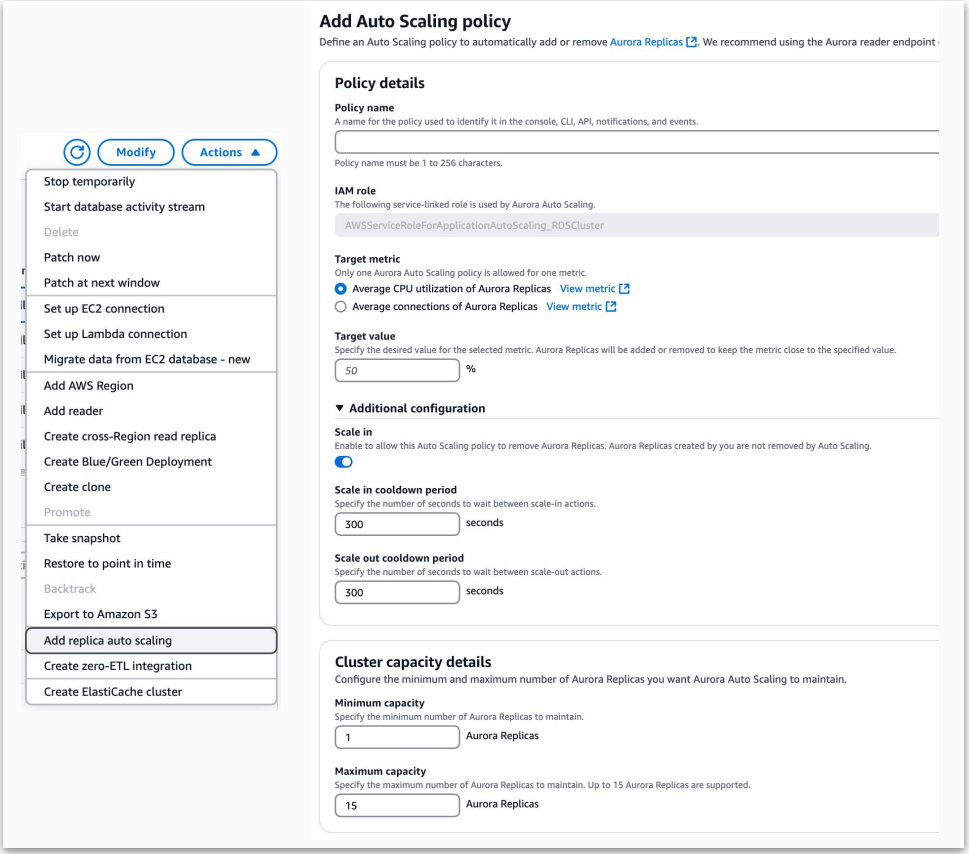
Horizontal Scaling

- Scalability
- High Availability
- Shard Management
- Topology Management
- Monitoring
- Support for Transactions
- Query route



Autoscaling

- Aurora only feature
- Capacity
  - Min 0 replica
  - Max 15 replicas
- Target Metric
  - CPU Utilization
  - Average connections
- Cooldown period (default 5 min)



Security

🔑 IAM for access, store credentials in Secrets Manager

🔒 Principle of least privilege granting minimal permissions

🏠 Deploy the database and application in the same VPC

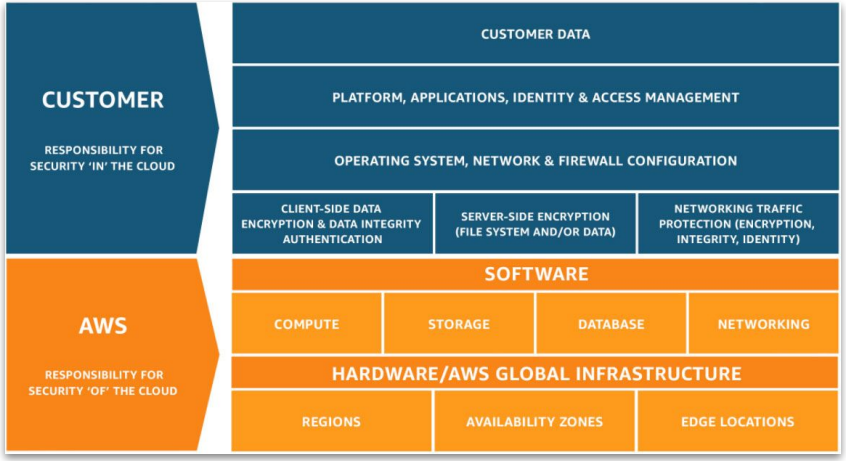
🌍 Don't expose the instance publicly, use private subnets and VPNs where necessary

🔐 Encryption at REST. Storage encryption to protect sensitive data. Use TLS/SSL for in-transit encryption

📲 Multi-factor authentication where possible

📅 Track CVEs and keep the database on the latest minor version

🔍 Enable and configure audit plugin where possible



Prompt engineering

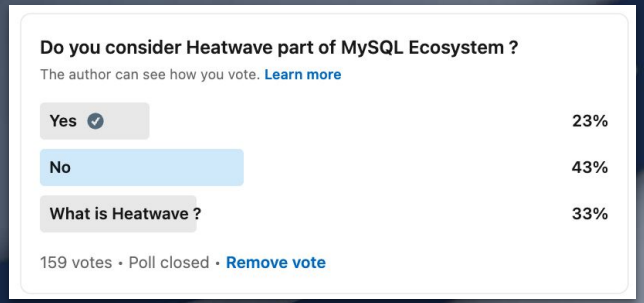
create SQL code to insert 10000 rows in below mysql table

```
CREATE TABLE t1 (
  id int unsigned NOT NULL AUTO_INCREMENT,
  p1 int DEFAULT NULL,
  pp1 int DEFAULT NULL,
  c1 int DEFAULT NULL,
  v1 int NOT NULL,
  name varchar(100) DEFAULT NULL,
  zzzz int DEFAULT NULL,
  tt int DEFAULT NULL,
  PRIMARY KEY (id)
) ENGINE=InnoDB
```

```
INSERT INTO `t1` (`p1`, `pp1`, `c1`, `v1`, `name`, `zzzz`, `tt`)
SELECT
    FLOOR(RAND() * 1000),   -- Random integer for p1
    FLOOR(RAND() * 1000),   -- Random integer for pp1
    FLOOR(RAND() * 1000),   -- Random integer for c1
    FLOOR(RAND() * 5000) + 1, -- v1 must be NOT NULL
    CONCAT('Name_', FLOOR(RAND() * 10000)), -- Random name
    FLOOR(RAND() * 1000),   -- Random integer for zzzz
    FLOOR(RAND() * 1000)    -- Random integer for tt
FROM (SELECT 1 UNION ALL SELECT 2 UNION ALL SELECT 3 UNION ALL SELECT 4
UNION ALL
      SELECT 5 UNION ALL SELECT 6 UNION ALL SELECT 7 UNION ALL SELECT 8
UNION ALL
      SELECT 9 UNION ALL SELECT 10) AS a
CROSS JOIN (SELECT 1 AS b FROM information_schema.tables LIMIT 1000) AS b;
```

Cloud Only

- New MySQL Hypergraph Optimizer - OCI Heatwave
- Vector Store and GEN AI - OCI Heatwave
- MySQl Autopilot Indexing - OCI Heatwave
- Javascript Stored Programs Support - OCI Heatwave
- Aurora features (parallel query, ML) - AWS
- Aurora Global Database - AWS
- Vector Store - GCP
- Adaptive Caching Layer - GCP



Thank You!

Questions?

Pythian

love your data