### **Migrating from AWS to GCP for a Retail E-Commerce Platform**

#### **Problem Statement**

A retail e-commerce company, **ShopEase**, currently hosts its database on **Amazon RDS for MySQL** in AWS. The database contains critical data such as customer information, product catalogs, orders, and payment details. Due to **cost optimization** and the need for better integration with GCP's data analytics tools (BigQuery, Looker), the company has decided to migrate its database to **Google Cloud SQL for MySQL**.

The objective of Migration is to:

1. **Minimal downtime** to avoid disruption to online sales.
2. **Data integrity** and consistency.
3. **Compatibility** of database objects (tables, stored procedures, functions, etc.).
4. **Scalability** to handle future growth.
5. **Cost optimization** to reduce operational expenses.

#### **Zettabolt , as migration partner did following things-**

#### Started analyzing the Sample Data and identified in-scope and out-scope objects.

#### Identified best suited Migration Approach based on the sample data

#### Created a robust Implementation plan and classified the whole migration process into 6 crucial phases /Milestones.

#### Performed Cost Optimization Strategies during and post Migration.

#### Overcome the challenges and hurdles during the migration

#### Let's Go deeper and see the whole migration plan-

#### **Sample Data**

The database contains the following tables:

1. **Customers**
   * customer\_id (Primary Key)
   * name
   * email
   * address
   * created\_at
2. **Products**
   * product\_id (Primary Key)
   * name
   * price
   * category
   * stock\_quantity
3. **Orders**
   * order\_id (Primary Key)
   * customer\_id (Foreign Key)
   * order\_date
   * total\_amount
4. **Order\_Items**
   * order\_item\_id (Primary Key)
   * order\_id (Foreign Key)
   * product\_id (Foreign Key)
   * quantity
   * price
5. **Payments**
   * payment\_id (Primary Key)
   * order\_id (Foreign Key)
   * payment\_method
   * amount
   * payment\_date

#### **Size of Data**

* **Total Database Size**: 4 TB (including indexes and logs).
* **Breakdown**:
  + Customers: 500 GB (25 million records).
  + Products: 100 GB (1 million records).
  + Orders: 800 GB (50 million records).
  + Order\_Items: 500 GB (200 million records).
  + Payments: 100 GB (50 million records).

#### **Migration Approach**

##### **1. Assessment and Planning**

* **Inventory**: Identify all tables, stored procedures, and functions in the AWS RDS MySQL database.
* **Compatibility Check**: Verify that all MySQL features used in AWS RDS are supported in GCP Cloud SQL.
* **Migration Strategy**: Use **AWS Database Migration Service (DMS)** for continuous replication to minimize downtime.

##### **2. Schema Migration**

* Export the schema from AWS RDS using mysqldump or AWS Schema Conversion Tool (SCT).
* Adjust the schema for GCP compatibility (e.g., remove AWS-specific extensions).
* Import the schema into GCP Cloud SQL.

##### **3. Data Migration**

* Use **AWS DMS** to replicate data from AWS RDS to GCP Cloud SQL.
* Perform an initial bulk load followed by continuous replication for real-time sync.
* Validate data integrity using row counts and checksums.

##### **4. Stored Procedures and Functions**

* Export stored procedures and functions from AWS RDS.
* Refactor any AWS-specific SQL code to be compatible with GCP Cloud SQL.
* Import the refactored code into GCP.

##### **5. Cutover and Go-Live**

* Stop writes to the AWS database during the final sync.
* Redirect the application to the GCP Cloud SQL database.
* Monitor performance and resolve any issues.

#### **Challenges and Fixes**

| **Challenge** | **Fix** |
| --- | --- |
| **Large Data Volume** | Use chunked migration and parallel processing. |
| **Network Latency** | Use AWS Direct Connect and GCP Cloud Interconnect for high-speed data transfer. |
| **Downtime** | Use AWS DMS for continuous replication to minimize downtime. |
| **Compatibility Issues** | Refactor AWS-specific SQL code and test thoroughly. |
| **Data Integrity** | Perform pre- and post-migration validation using checksums and sample queries. |

#### 

#### **Implementation Plan**

1. **Phase 1: Pre-Migration**
   * Inventory database objects.
   * Analyze compatibility and plan migration strategy.
   * Set up GCP Cloud SQL instance.
2. **Phase 2: Schema Migration**
   * Export schema from AWS RDS.
   * Adjust schema for GCP compatibility.
   * Import schema into GCP Cloud SQL.
3. **Phase 3: Data Migration**
   * Use AWS DMS for initial bulk load and continuous replication.
   * Validate data integrity.
4. **Phase 4: Code Migration**
   * Export stored procedures and functions from AWS RDS.
   * Refactor and import into GCP Cloud SQL.
5. **Phase 5: Cutover**
   * Stop writes to AWS RDS during the final sync.
   * Redirect application to GCP Cloud SQL.
6. **Phase 6: Post-Migration**
   * Monitor performance and resolve issues.
   * Optimize the database for GCP.

#### **Tools and Applications Involved**

##### **AWS Tools**

1. **Amazon RDS**: Hosts the MySQL database.
2. **AWS Database Migration Service (DMS)**: For continuous data replication.
3. **AWS Schema Conversion Tool (SCT)**: For schema and code conversion.
4. **AWS Direct Connect**: For high-speed, secure connectivity between AWS and GCP.

##### **GCP Tools**

1. **Google Cloud SQL**: Target MySQL database.
2. **BigQuery**: For analytics and reporting post-migration.
3. **Cloud Monitoring**: For performance monitoring and troubleshooting.
4. **Cloud Interconnect**: For high-speed connectivity between AWS and GCP.

##### **Third-Party Tools**

1. **Striim**: For real-time data integration and migration.
2. **Fivetran**: For automated data pipeline creation.
3. **Apache Airflow**: For orchestrating ETL workflows.
4. **dbt (Data Build Tool)**: For data validation and testing.

##### **Open-Source Tools**

1. **mysqldump**: For exporting schema and data from AWS RDS.
2. **mysql**: For importing schema and data into GCP Cloud SQL.
3. **pgloader**: For data migration (if PostgreSQL is involved).

#### **Cost Optimization Strategies**

1. **Right-Sizing Resources**
   * Analyze current AWS RDS instance usage (CPU, memory, storage).
   * Choose a GCP Cloud SQL instance type that matches the workload (e.g., db-n1-standard-4 instead of over-provisioned resources).
   * Use **GCP’s Committed Use Discounts** for long-term cost savings.
2. **Storage Optimization**
   * Use **GCP’s Standard or SSD storage** based on performance needs.
   * Enable **automatic storage scaling** to avoid over-provisioning.
3. **Network Cost Reduction**
   * Use **GCP’s Premium Tier Network** for predictable pricing and low latency.
   * Leverage **AWS Direct Connect** and **GCP Cloud Interconnect** to reduce data transfer costs between AWS and GCP.
4. **Serverless and Managed Services**
   * Use **GCP BigQuery** for analytics instead of running expensive ETL pipelines on AWS.
   * Leverage **Cloud SQL’s automated backups** and maintenance to reduce operational overhead.
5. **Monitoring and Cost Management**
   * Use **GCP’s Cost Management Tools** to monitor and optimize spending.
   * Set up **budget alerts** to avoid unexpected costs.
6. **Data Lifecycle Management**
   * Archive historical data to **GCP Coldline Storage** for cost-effective long-term storage.
   * Use **BigQuery partitioning** to reduce query costs for large datasets.

#### **Cost Benefits Post-Migration**

| **Cost Component** | **AWS Cost (Monthly)** | **GCP Cost (Monthly)** | **Savings (%)** |
| --- | --- | --- | --- |
| **Database Instance** | **$1,500 (db.m5.xlarge)** | **$1,000 (db-n1-standard-4)** | **33%** |
| **Storage** | **$300 (500 GB SSD)** | **$200 (500 GB SSD)** | **33%** |
| **Backups** | **$100** | **$50 (automated backups)** | **50%** |
| **Data Transfer** | **$200 (AWS to Internet)** | **$100 (GCP Premium Tier)** | **50%** |
| **Analytics (BigQuery)** | **$500 (AWS Redshift)** | **$300 (BigQuery)** | **40%** |
| **Total** | **$2,600** | **$1,650** | **36.5%** |

#### 

#### **Conclusion**

By following a structured approach and leveraging tools like **AWS DMS**, **GCP Cloud SQL**, and third-party applications like **Striim** and **Fivetran**, **ShopEase** successfully migrated its database from AWS to GCP with minimal downtime and ensured data integrity. The migration enabled the company to integrate seamlessly with GCP's analytics tools, paving the way for data-driven decision-making. Additionally, cost optimization strategies such as right-sizing resources, storage optimization, and leveraging managed services helped **ShopEase** reduce operational expenses significantly, achieving **36.5% in monthly savings**.