Available online www.jsaer.com

***Journal of Scientific and Engineering Research*, 2022, 9(11):358-363**

Research Article

ISSN: 2394-2630
CODEN(USA): JSERBR

Migration of On-Premise Databases to Amazon RDS – Challenges and Solutions

Mohit Thodupunuri

MS in Computer Science,
Sr Software Developer - Charter Communications Inc.
Email id: Mohit.thodupunuri@gmail.com

Abstract: Migrating on-premises databases to Amazon Relational Database Service (RDS) offers organizations benefits such as enhanced scalability, automated maintenance, and managed infrastructure. Key features of Amazon RDS include Multi-AZ deployments, read replicas, and integration with AWS Identity and Access Management (IAM). These features reduce operational overhead through automated patching and elastic storage scaling. However, challenges such as schema incompatibility, network latency, transactional consistency during cutover, and security policy alignment can arise during migration. Compared to lift-and-shift approaches, RDS migrations require meticulous planning using AWS Database Migration Service (DMS) and Schema Conversion Tool (SCT). This paper examines technical hurdles—such as downtime minimization and data integrity—and proposes solutions like parallel batch processing and Virtual Private Cloud (VPC) peering.

Keywords: Database Migration, Amazon RDS, AWS DMS, Schema Conversion Tool, Multi-AZ, Transactional Consistency, VPC Peering, CloudWatch Metrics

1. Introduction

Organizations seek to enhance their database infrastructures by migrating from on-premises environments to cloud-based solutions. Amazon Relational Database Service (RDS) is a managed service that supports various database engines, including MySQL, PostgreSQL, and Oracle. It abstracts hardware provisioning, automates failover, and provides point-in-time recovery, thereby reducing the complexities associated with database management. [1]

On-premises databases often rely on physical servers and static storage area networks (SANs), which lack the elasticity and scalability offered by cloud services. Migrating to Amazon RDS replaces manual replication topologies with read replicas synchronized using asynchronous log-shipping, enhancing data availability and performance. [2]

Amazon RDS is engineered for high availability through Multi-AZ deployments, which replicate databases across multiple Availability Zones using synchronous block-level replication [3]. The service integrates with AWS Key Management Service (KMS) for encryption at rest and AWS Identity and Access Management (IAM) for granular access controls. However, migration workflows often require schema refactoring to address unsupported features, such as custom PL/SQL extensions. AWS Database Migration Service (DMS) employs change data capture (CDC) to replicate ongoing transactions with minimal latency, thereby minimizing downtime during migration.

2. Literature Review

Database migration to cloud environments, particularly Amazon RDS, is a well-researched topic with various strategies and challenges. Researchers have explored methodologies for migrating mission-critical databases, addressing data security concerns, and optimizing performance post-migration. This section reviews relevant literature to highlight key insights into cloud database migration.

Narani et al. (2018) [1] examined strategies for migrating large, mission-critical databases to the cloud. They emphasized incremental data transfer methods to minimize downtime. Similarly, Lakshmi (2018) [4] discussed the benefits of schema conversion tools to handle database structure modifications. Both studies underscored the importance of thorough pre-migration assessments to identify compatibility issues before execution.

Security remains a major concern when shifting to cloud-based relational databases. Dahshan (2014) [2] analyzed data security challenges in cloud storage, focusing on encryption mechanisms and role-based access controls. Armstrong (2020) [5] expanded on these concerns, providing security best practices for AWS migrations. His work suggested implementing IAM policies and VPC security groups to restrict unauthorized access. Seenivasan (2021) [6] reinforced these findings by highlighting the need for data governance frameworks to maintain compliance during cloud adoption.

Performance optimization is another key factor in successful database migration. Ellison (2017) [7] evaluated cloud migration options for relational databases, emphasizing the role of query optimization techniques post-migration. Similarly, Dombrovskaya et al. (2021) [9] explored PostgreSQL query tuning in cloud environments, advocating for indexing strategies and execution plan analysis. These studies confirmed that fine-tuning database configurations ensures efficient performance after migrating to Amazon RDS.

Finally, AWS's official documentation [3] detailed Amazon RDS Read Replicas, which enhance database scalability and load balancing. Andrews (2018) [8] further explored data transfer methods, particularly AWS Snowball Edge, for migrating large datasets with minimal latency. Their findings align with industry best practices, demonstrating how hybrid migration approaches can improve efficiency.

3. Problem Statement: Downtime, Data Incompatibility, Latency Issues, & Security Policy

Migrating on-premises databases to Amazon Relational Database Service (RDS) introduces several technical challenges. While organizations benefit from managed infrastructure, automated scaling, and cost efficiencies, the transition demands meticulous planning. Key concerns include downtime during cutover, schema incompatibility, network performance, and security policy alignment. Addressing these challenges requires a deep understanding of database replication, AWS networking, and access control mechanisms. [4]

Downtime Management During Cutover

Legacy database systems rely on synchronous transaction processing, making downtime a critical bottleneck. During migration, final data synchronization requires temporary service interruptions, disrupting ongoing Online Transaction Processing (OLTP) workloads. Organizations using high-availability architectures with active-active clustering face additional challenges, as transactional consistency must be preserved while switching over to RDS. [4] [5]

To minimize downtime, AWS Database Migration Service (DMS) enables Change Data Capture (CDC), replicating real-time modifications from the source database. However, high transaction rates can overwhelm the migration pipeline, leading to lag in synchronization. Parallel processing mitigates this issue, but it demands sufficient compute resources in the replication instance. Additionally, configuring the cutover window requires balancing minimal downtime with ensuring full data consistency.[7]

Schema and Data Type Incompatibility

On-premises databases often use proprietary schema structures, complex stored procedures, and unsupported data types. For example, Oracle's RAW and LONG data types lack direct compatibility with PostgreSQL on RDS, necessitating conversion. Similarly, SQL Server's TEXT and NTEXT types require migration to VARCHAR(MAX), which alters storage behavior.

AWS Schema Conversion Tool (SCT) helps automate these changes, but manual intervention remains necessary for custom stored procedures and triggers. Complex joins, recursive queries, and platform-specific indexing strategies must be re-engineered to maintain query performance post-migration. Without proper schema optimization, workloads may experience degraded performance and increased query execution times. [4]

Network Latency and Bandwidth Constraints

Transferring terabytes of database records over the internet introduces bandwidth bottlenecks and latency spikes. [7]

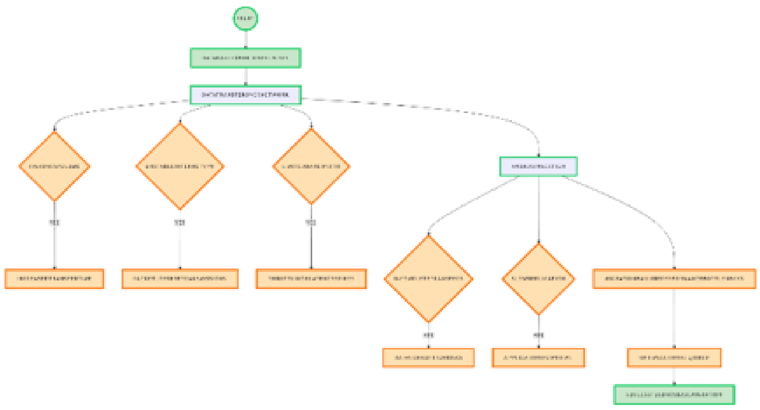


Figure 1: Network latency and bandwidth constraints affecting on-premises database migration to AWS RDS.

The flowchart shows how network latency and bandwidth limitations impact database migration from an on-premises environment to Amazon RDS. During the data transfer phase, large volumes of records must traverse either the public internet or a dedicated VPN/Direct Connect link. If bandwidth is insufficient or the network experiences instability, packet loss and retransmissions occur.

These inefficiencies prolong transfer times, cause replication delays, and may even lead to migration failures. Optimizing the process through parallel uploads, data compression, or AWS Snowball helps mitigate these issues, ensuring a successful migration with minimal downtime.[4]

Packet loss and TCP retransmissions further complicate bulk migrations, especially for databases with high write throughput. To address these issues, AWS Snowball provides a physical data transport solution, reducing the dependency on live network transfers. Alternatively, parallelization and compression techniques—such as MySQL’s mysqldump with gzip compression—reduce data transfer times over limited bandwidth connections. [4] [7]

Security Policy Misalignment

On-premises databases enforce access control using firewalls, local user roles, and network segmentation. Transitioning to RDS requires reconfiguring these security models within AWS Identity and Access Management (IAM) and Security Groups. [4]

Legacy Role-Based Access Control (RBAC) systems may not map directly to AWS’s managed database authentication. For example, SQL Server’s Windows Authentication does not work natively in RDS, requiring a shift to IAM-based authentication or native SQL logins. Firewall rules must also be updated to allow application servers to communicate with the RDS instance over private IPs, ensuring compliance with organizational security policies.[6]

Without proper alignment, misconfigured access policies may lead to unauthorized access, increased attack surfaces, or service disruptions. Implementing AWS Key Management Service (KMS) for encryption and AWS CloudTrail for auditing mitigates these risks, ensuring data security remains intact throughout migration.

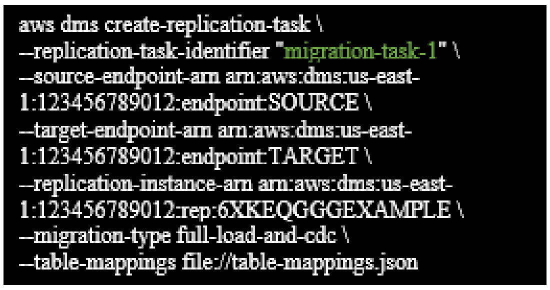
4. Solution: Optimizing Migration for Reliability and Performance

Migrating on-premise databases to Amazon RDS requires a structured approach. Addressing downtime, data integrity, schema compatibility, network bottlenecks, and security policies ensures a seamless transition. The following solutions use AWS services to mitigate these challenges effectively.

Near-Zero Downtime with AWS DMS and CDC

AWS Database Migration Service (DMS) minimizes downtime by using Change Data Capture (CDC). CDC continuously replicates live transactions from the source database to RDS, preventing long service interruptions. Binary log coordinates, such as MySQL binlog and Oracle Redo Log, track changes and synchronize updates with subsecond latency.

The following AWS CLI command sets up a DMS replication task with CDC:



```
aws dms create-replication-task \
–replication-task-identifier "migration-task-1" \
–source-endpoint-arn arn:aws:dms:us-east-
1:123456789012:endpoint:SOURCE \
–target-endpoint-arn arn:aws:dms:us-east-
1:123456789012:endpoint:TARGET \
–replication-instance-arn arn:aws:dms:us-east-
1:123456789012:rep:6XKEQGGGEXAMPLE \
–migration-type full-load-and-cdc \
–table-mappings file://table-mappings.json
```

Figure 2: AWS CLI command to create a DMS task with CDC.

This script configures a migration task that performs an initial full-load migration followed by CDC replication. By continuously syncing changes, AWS DMS ensures minimal disruption during cutover. Scheduling the cutover during off-peak hours further reduces impact.[3]

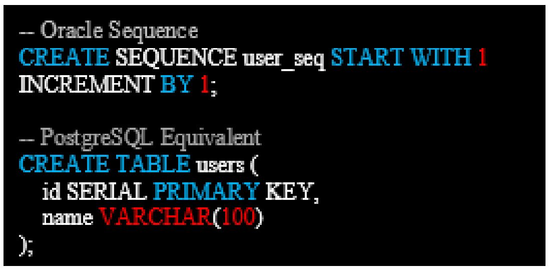
Schema Refactoring Using AWS SCT

On-premise databases often use proprietary data types and stored procedures unsupported by RDS. The AWS Schema Conversion Tool (SCT) converts these constructs into compatible formats. For example, Oracle's RAW type maps to PostgreSQL's BYTEA, and Oracle sequences convert to PostgreSQL GENERATED AS IDENTITY.

To use AWS SCT for Schema Conversion:

1. Install AWS SCT and connect to the source database.
2. Analyze schema incompatibilities and generate conversion recommendations.
3. Apply transformations to modify schema definitions.
4. Deploy the converted schema to Amazon RDS.

The following SQL script shows an Oracle sequence conversion:



```
-- Oracle Sequence
CREATE SEQUENCE user_seq START WITH 1
INCREMENT BY 1;

-- PostgreSQL Equivalent
CREATE TABLE users (
   id SERIAL PRIMARY KEY,
   name VARCHAR(100)
);
```

Figure 3: Oracle sequence to PostgreSQL identity column conversion.

AWS SCT automates most conversions but may require manual intervention for complex stored procedures. Using AWS Glue, large binary objects (BLOBs) can be transformed into RDS-compatible formats before migration.

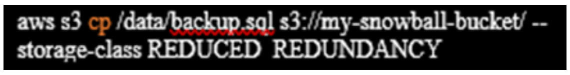
Accelerated Data Transfer via Snowball Edge

For databases exceeding terabytes, network-based transfers become impractical. AWS Snowball Edge enables offline data migration by physically shipping encrypted storage devices to AWS data centers [4].

Typically, a Snowball workflow for bulk data transfer includes [9]:

1. Order a Snowball Edge device from the AWS console.
2. Export the database and compress data using LZ4 to reduce size.
3. Copy data onto the Snowball device using the AWS Snowball client.
4. Ship the device back to AWS, where data is uploaded to S3.
5. Import the data into RDS using AWS Glue or custom scripts

The data copy command is listed in figure 4.



```
aws s3 cp /data/backup.sql s3://my-snowball-bucket/ --
storage-class REDUCED_REDUNDANCY
```

Figure 4: Copying database backups to an AWS Snowball Edge device.

Using multipart uploads, Snowball Edge ensures parallel processing for faster transfers. After ingestion, AWS DMS can synchronize delta changes before final cutover.

Security Harmonization with VPC Peering and IAM

AWS security policies differ from on-premise models, requiring alignment to prevent unauthorized access. Virtual Private Cloud (VPC) peering allows secure communication between an on-premise data center and Amazon RDS without exposing data to the internet.

Configuring a VPC Peering for secure migration includes the following steps:

1. Create a VPC peering connection using the command aws ec2 create-vpc-peering-connection --vpc-id vpc-12345678 --peer-vpc-id vpc-87654321
2. Modify route tables to allow traffic between networks using the command aws ec2 create-route --route-table-id rtb-12345678 --destination-cidr-block 192.168.0.0/16 --vpc-peering-connection-id pcx-12345678

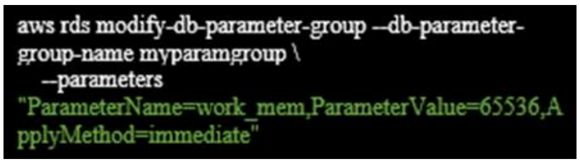
To enhance security, IAM roles must follow the principle of least privilege. Encrypting data in transit with TLS 1.3 ensures secure communication. AWS Config audits security policies to enforce compliance.

Query Performance Tuning with RDS Parameter Groups

After migration, performance optimization ensures efficient database operations. AWS RDS parameter groups allow fine-tuning database configurations.

To optimize PostgreSQL with parameter groups [9]:

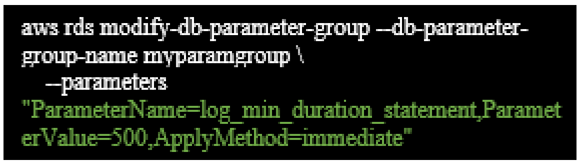
1. Increase memory allocation for query caching using the command shown in figure 5.



```
aws rds modify-db-parameter-group –db-parameter-
group-name myparamgroup \
   –parameters
"ParameterName=work_mem,ParameterValue=65536,A
pplyMethod=immediate"
```

Figure 5: Increasing memory allocation.

2. Enable query logging for performance analysis using the command in figure 6.



```
aws rds modify-db-parameter-group –db-parameter-
group-name myparamgroup \
   –parameters
"ParameterName=log_min_duration_statement,Paramet
erValue=500,ApplyMethod=immediate"
```

Figure 6: Configuring RDS parameter groups for performance optimization.

Tuning parameters like innodb_buffer_pool_size and enabling Amazon ElastiCache, read-heavy workloads can be offloaded, improving database responsiveness.

Migrating to Amazon RDS demands careful planning. AWS DMS with CDC ensures near-zero downtime. AWS SCT modernizes schemas for cloud compatibility. Snowball Edge accelerates large-scale migrations. VPC peering and IAM enforce security, while RDS parameter tuning optimizes performance. These solutions collectively ensure a seamless, secure, and efficient transition to cloud-based database management.

5. Analysis and Recommendations

Migrating on-premise databases to Amazon RDS requires careful analysis of key factors, including downtime mitigation, data integrity, cost optimization, and compliance. Each area presents unique challenges that organizations must address to ensure a smooth transition.

Minimizing downtime is crucial for business continuity. A blue/green deployment strategy helps achieve near-zero downtime by running a parallel RDS instance, synchronizing data using AWS DMS CDC, and switching traffic once validation is complete. Organizations must also monitor CDC latency using Amazon CloudWatch to prevent replication lag that could delay the final cutover.

Ensuring data accuracy during migration prevents application failures and inconsistencies. Organizations must validate data before switching to the new database. Tools like pg_checksums (PostgreSQL) and mysqldbcompare (MySQL) help compare source and target databases, ensuring no corruption occurs during the migration.

Cloud database costs can escalate if not managed efficiently. Choosing the right pricing model ensures affordability without sacrificing performance. Reserved Instances provide savings for steady workloads, while Aurora Serverless scales dynamically based on traffic, reducing costs for variable demand applications.

Regulatory compliance and security measures must be enforced when migrating to the cloud. Organizations should use AWS CloudTrail for audit logging and implement Network ACLs (NACLs) to restrict unauthorized access. IAM policies with least-privilege access further enhance security by limiting database permissions.

Following are our recommendations based on our analysis.

• Use blue/green deployments and AWS DMS CDC to replicate transactions before switching traffic.

• Run checksum validation with pg_checksums (PostgreSQL) or mysqldbcompare (MySQL) before final cutover.

• Deploy Reserved Instances for predictable workloads and Aurora Serverless for fluctuating demand.

• Enable AWS CloudTrail for logging and restrict access using Network ACLs and least-privilege IAM roles.

6. Conclusion

Migrating on-premise databases to Amazon RDS offers significant advantages, including improved scalability, automated management, and enhanced security. However, the process involves complex challenges such as downtime management, data consistency, network constraints, and compliance alignment. Organizations must carefully plan their migration strategy to minimize disruptions and ensure data integrity.

With the help of AWS DMS with CDC, businesses can reduce downtime while maintaining transactional consistency. Schema refactoring using AWS SCT helps resolve compatibility issues, while accelerated data transfer methods like AWS Snowball Edge address network limitations. Additionally, security best practices such as VPC peering, IAM role management, and AWS CloudTrail auditing ensure compliance and data protection.

Ultimately, a well-structured migration strategy that integrates cost optimization, performance tuning, and security enforcement enables organizations to fully capitalize on Amazon RDS. As cloud adoption continues to grow, refining migration methodologies and using AWS-native tools will be critical for seamless, efficient database modernization.

References

[1]. Narani, S. R., Ayyalasomayajula, M. M. T., & Chintala, S. (2018). Strategies For Migrating Large, Mission-Critical Database Workloads To The Cloud. Webology (ISSN: 1735-188X), 15(1).

[2]. Dahshan, M. M. (2014). Data security in cloud storage services.

[3]. “Amazon RDS Read Replicas | Cloud Relational Database | Amazon Web Services,” (2014) Amazon Web Services, Inc. https://aws.amazon.com/rds/features/read-replicas/

[4]. Lakshmi, N. G. (2018). Database Migration on Premises to AWS RDS. EAI Endorsed Transactions on Cloud Systems, 3(11).

[5]. Armstrong, J. (2020). Migrating to AWS: A Manager's Guide: how to Foster Agility, Reduce Costs, and Bring a Competitive Edge to Your Business. O'Reilly Media.

[6]. Seenivasan, D. (2021). Transforming Data Warehousing: Strategic Approaches and Challenges in Migrating from On-Premises to Cloud Environments.

[7]. Ellison, M. (2017). Evaluating Cloud Migration Options for Relational Databases (Doctoral dissertation, University of York).

[8]. Andrews, P. C. (2018). “Putting It Together, That’s What Counts”: Data Foam, a Snowball and Researcher Evaluation. Humans and Machines at Work: Monitoring, Surveillance and Automation in Contemporary Capitalism, 203-229.

[9]. Dombrovskaya, H., Novikov, B., & Bailliekova, A. (2021). PostgreSQL Query Optimization. Apress.