AWS Database Blog

Migrating SQL Server to Amazon RDS using native backup and restore

by Veerendra Nayak and Sam Selvan | on 15 DEC 2020 | in RDS For SQL Server | Permalink | Comments | Share

Running Microsoft SQL Server on Amazon Relational Database Service (Amazon RDS) automates time-consuming administration tasks such as installation, disk provisioning and management, patching, minor and major version upgrades, and backup and recovery of your SQL Server databases. Amazon RDS also offers high availability using a Multi-AZ solution, replicating data synchronously across different Availability Zones.

Currently, you can migrate self-managed SQL server to Amazon RDS for SQL Server using any of the following methods:

- Using AWS Database Migration Service (AWS DMS). For more information, see Migrating your SQL Server database to Amazon RDS for SQL Server using AWS DMS
- Using SQL Server transactional replication. For more information, see migrate to Amazon RDS for SQL Server using transactional replication
- Using SQL Server backup and restore features

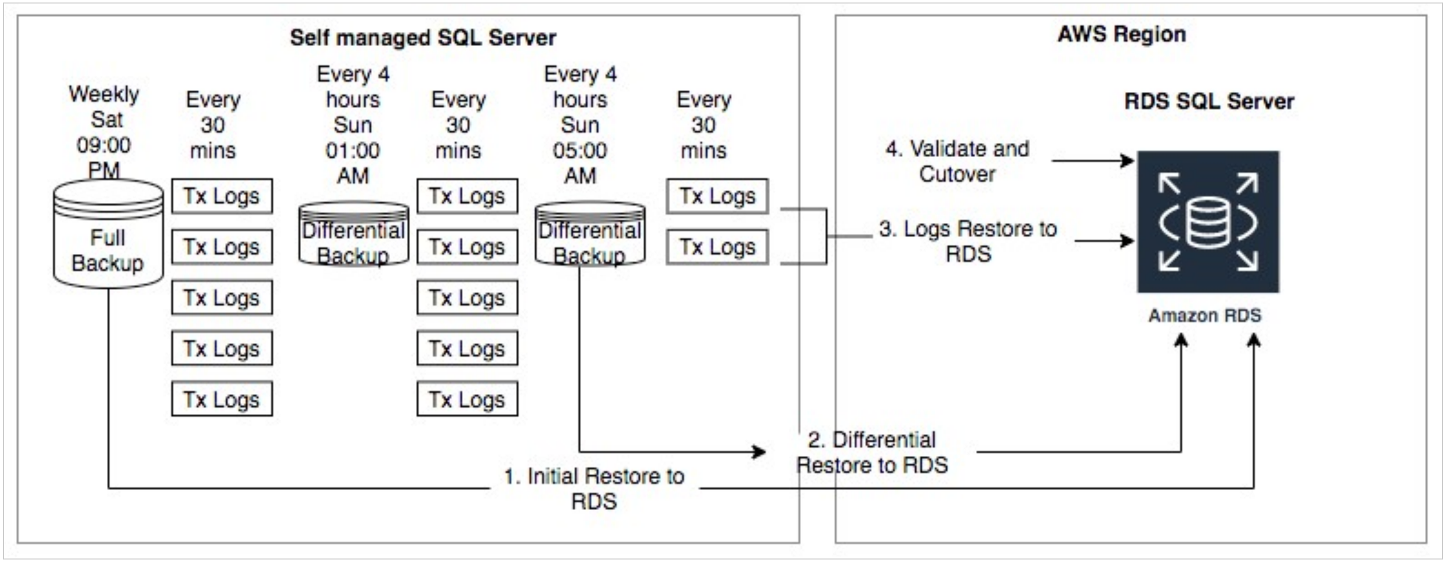
In this post, we demonstrate how to use the SQL Server backup and restore features to migrate to Amazon RDS for SQL Server. This involves performing a full backup followed by differential and log backups on the source instance and restoring the same backups to the target RDS for SQL Server instance. With the help of differential and log backup, we significantly reduce application cutover time during the migration process.

On Amazon RDS for SQL Server, the native backup and restore functionality is supported with the help of stored procedures.

For this post, we use a self-managed SQL Server database that runs on the following backup schedule:

- Full backup every Saturday at 9:00 PM
- Differential backup every 4 hours
- Transaction log backup every 30 mins

The following diagram illustrates this architecture.



The migration process can be described with the following high-level steps:

1. Restore a full backup to Amazon RDS for SQL Server.
2. Restore a differential backup to Amazon RDS for SQL Server. Because differential backups are cumulative, we restore the most recent differential backup since the last full backup.
3. Restore subsequent transaction log backups to the latest point in time.
4. Validate and cut over.

As of this writing, differential and log restores are not supported on Multi-AZ instances. We suggest creating a single Availability Zone RDS instance for migration purposes and modifying it to Multi-AZ at a later point.

Restoring a full backup to Amazon RDS for SQL Server

The backup files need to be in Amazon Simple Storage Service (Amazon S3) so they can be restored on Amazon RDS for SQL Server. To copy the files to Amazon S3, use the Amazon S3 console or the AWS Command Line Interface (AWS CLI).

When the files are available on Amazon S3, you create an AWS Identity and Access Management (IAM) role to provide Amazon RDS access to the S3 bucket with the backup files.

Creating an IAM policy and role

We have included a sample policy in this post for reference. Create a new role with this policy and add trust for Amazon RDS to assume this role.

```
{
"Version": "2012-10-17",
"Statement":
```

```
[
{
"Effect": "Allow",
"Action":
[ "s3:ListBucket", "s3:GetBucketLocation"
],
"Resource": "arn:aws:s3:::bucket_name"
},
{
"Effect": "Allow",
"Action":
[ "s3:GetObject", "s3:PutObject", "s3:ListMultipartUploadParts", "s3:AbortMultipartU
],
"Resource": "arn:aws:s3:::bucket_name/*"
}
```

Adding trust for Amazon RDS to assume the role

When you create an IAM role, you attach a trust relationship and a permissions policy. The trust relationship allows Amazon RDS to assume this role. See the following code:

```
{
"Version": "2012-10-17",
"Statement":
[{
"Effect": "Allow",
"Principal": {"Service": "rds.amazonaws.com"},
"Action": "sts:AssumeRole"
}]
}
```

Adding an option group for backup and restore

Option groups in Amazon RDS provide flexibility to manage additional features. In this post, we add an option for native backup and restore. For a full list of available options, see Options for the Microsoft SQL Server database engine.

You can create a new option group if it doesn’t exist and then add an option `SQLSERVER_BACKUP_RESTORE` to that option group. Complete the following steps:

1. On the Amazon RDS console, choose **Option Groups** in the navigation pane.
2. Choose **Create Group**.
3. Enter the name, description, engine, and engine version of your server.

4. Choose **Create**.
5. Select the option group that you created, and choose **Add Option**.
6. Choose **SQLSERVER_BACKUP_RESTORE**.
7. Choose the IAM Role created in the earlier step.
8. Choose **Immediately**.
9. Choose **Add Option**.
10. Associate the option group with the DB instance by choosing **Databases** in the navigation pane.
11. Choose the target RDS SQL Server instance
12. On the **Actions** menu, choose **Modify**.
13. Under **Database Options**, choose the option group that you created.
14. Choose **Continue**.
15. Choose **Apply Immediately**.
16. Review the information and choose **Modify DB Instance**.

This option group modification has no downtime because an instance reboot is not required.

Restoring the full backup to Amazon RDS

After setting up the IAM role and option group, you can start the restore process. With the backup schedule scenario defined earlier, this is a backup taken on Saturday at 9:00 PM.

Amazon S3 has a size limit of 5 TB per file. For native backups of larger databases, you can use a multi-file backup on the source instance.

Run the following Amazon RDS restore procedures in SQL Server Management Studio:

```
use master
go

exec msdb.dbo.rds_restore_database
@restore_db_name='mydatabase',
@s3_arn_to_restore_from='arn:aws:s3:::bucketname/sqlserverbackups/mydb_full-1.bak',
@with_norecovery=1,
@type='FULL';
go
```

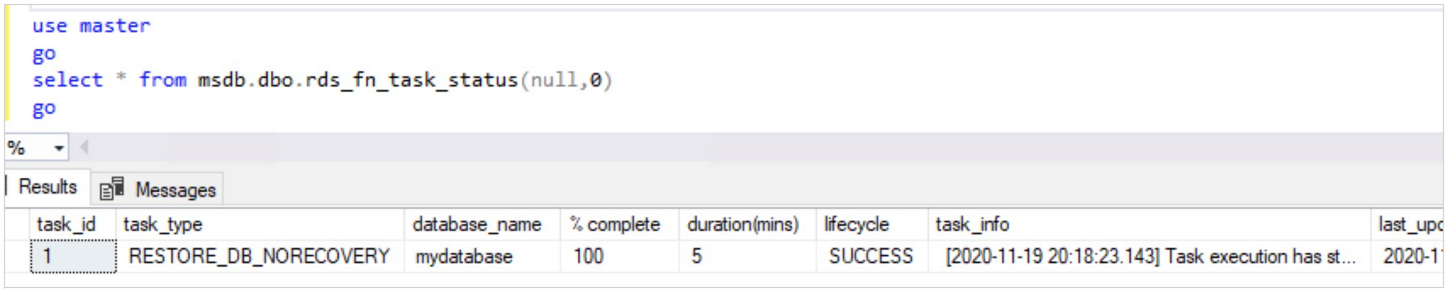
The preceding restore task uses the `NORECOVERY` option by specifying `@with_norecovery=1`. The database is left in the `RESTORING` state, allowing for subsequent differential or log restores.

You can check the status of the restore task with the following code:

```
use master
go
```

```
select * from msdb.dbo.rds_fn_task_status(null,0)
go
```

The restore process is complete when the lifecycle shows SUCCESS and percentage complete is 100.



```
use master
go
select * from msdb.dbo.rds_fn_task_status(null,0)
go
```

task_id	task_type	database_name	% complete	duration(mins)	lifecycle	task_info	last_upc
1	RESTORE_DB_NORECOVERY	mydatabase	100	5	SUCCESS	[2020-11-19 20:18:23.143] Task execution has st...	2020-1

Restoring differential backups to Amazon RDS for SQL Server

Differential backups have all the changes since the last full backup. Because differential backups are cumulative, use the latest differential backup file to restore. For our use case, we use a differential backup that was taken Sunday at 5:00 AM. See the following code:

```
use master
go
exec msdb.dbo.rds_restore_database
@restore_db_name='mydatabase',
@s3_arn_to_restore_from='arn:aws:s3:::bucketname/sqlserverbackups/mydb_diff-1.bak',
@type='DIFFERENTIAL',
@with_norecovery=1;
go
```

Restoring subsequent transaction log backups to Amazon RDS for SQL Server

Transaction logs record all the database modifications made by each transaction. You must create at least one full backup before you can create any log backups. The sequence of transaction log backups (called a *log chain*) is independent of data backups.

If you want to restore a database to a certain point, restore the full backup followed by the latest differential backup and all subsequent transaction logs up to that restore point.

For our use case, we restore the differential backup taken Sunday at 5:00 AM, so we need to restore all transaction log backups taken after 5:00 AM. We must apply log backups in the sequence in which they were created, without any gaps in the log chain.

To minimize the cutover time, you may want to consider taking one final manual log backup.

Restore all the log backups except the last one by specifying `@with_norecovery=1`:

```
use master
go
exec msdb.dbo.rds_restore_log
@restore_db_name='mydatabase',
@s3_arn_to_restore_from='arn:aws:s3:::bucketname/sqlserverbackup/mydb_log_n.trn',
@with_norecovery=1;
go
```

Before running a final log backup on the source database, issue a checkpoint to persist data pages located in the buffer pool to database files.

Also, while making a final log backup, update the status of the source database to be read-only to avoid any database changes on the source instance during cutover. Ensure there are no active transactions using `DBCC OPENTRAN`. See the following code:

```
-- source database
GO
ALTER DATABASE source_database SET SINGLE_USER WITH ROLLBACK IMMEDIATE
GO
ALTER DATABASE source_database SET READ_ONLY WITH ROLLBACK IMMEDIATE
GO
ALTER DATABASE source_database SET MULTI_USER
GO
```

When the database in read-only mode, we proceed with the final log backup on the source instance and restore it on Amazon RDS for SQL Server:

```
use master
go
exec msdb.dbo.rds_restore_log
@restore_db_name='mydatabase',
@s3_arn_to_restore_from='arn:aws:s3:::bucketname/sqlserverbackup/mydb_log_final.trn'
@with_norecovery=0;
go
```

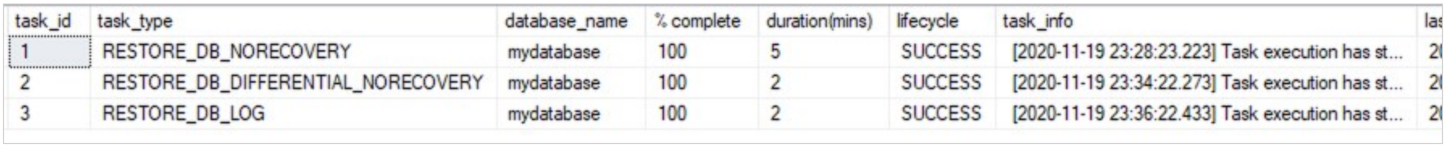
We restore the final log backup by specifying `@with_norecovery=0`. This brings the database state to online after the restore task completion.

Validating and cutting over

In the target RDS instance, check if all restore tasks completed successfully in Amazon RDS SQL for Server:

```
use master
go
select * from msdb.dbo.rds_fn_task_status(null,0)
go
```

The following screenshot shows the results.



task_id	task_type	database_name	% complete	duration(mins)	lifecycle	task_info	las
1	RESTORE_DB_NORECOVERY	mydatabase	100	5	SUCCESS	[2020-11-19 23:28:23.223] Task execution has st...	2
2	RESTORE_DB_DIFFERENTIAL_NORECOVERY	mydatabase	100	2	SUCCESS	[2020-11-19 23:34:22.273] Task execution has st...	2
3	RESTORE_DB_LOG	mydatabase	100	2	SUCCESS	[2020-11-19 23:36:22.433] Task execution has st...	2

Additionally, you can build in data validation to compare record count and data records between the source and target for key tables using linked servers. For instructions, see Implement Linked Servers with Amazon RDS for Microsoft SQL Server.

After successfully completing restore tasks and data validation, modify the RDS instance to enable Multi-AZ. When this is complete, update the endpoint in the application to point to the RDS for SQL Server instance.

This completes the migration process to Amazon RDS for SQL Server.

Summary

This post covered how to migrate self-managed SQL Server to Amazon RDS for SQL Server using native backup and restore functionality. With a successful migration to Amazon RDS, you can focus on what matters to your organization’s business, such as optimizing database design, capacity sizing, and performance tuning.

If you have any questions or suggestions, please leave us a comment.

About the Authors

Veerendra Nayak is a Senior Database Solution Architect with Amazon Web Services.

Sam Selvan is a Senior Database Solution Architect with Amazon Web Services.

TAGS: Amazon RDS for SQL Server