



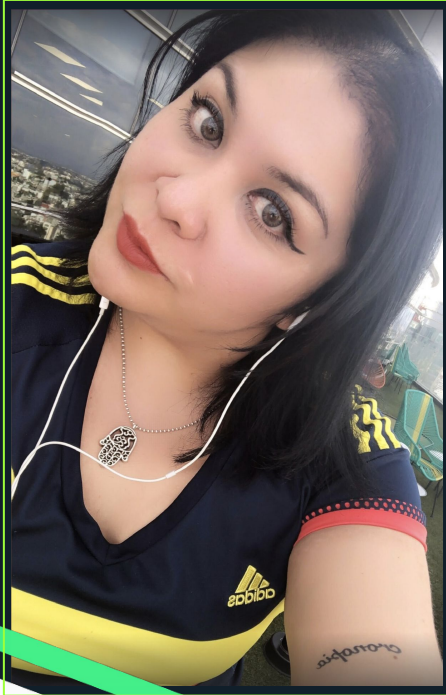
**Lo que se debe y no se debe hacer
para una actualización en
PostgreSQL AWS RDS**

Por: Joyce López

#Devday4w



About me



Joyce López

Database Administrator Sr Level 3

5.7 years @ Globant

Proud DBA

Open Source and NoSQL DBs Specialist.

Favorite cloud: AWS

Why we love RDS Databases?

Amazon Relational Database Service (RDS) allows users to set up, operate, and scale a database in the cloud. It provides cost-efficient and resizable capacity while automating time-consuming administration tasks such as hardware provisioning, setup, patching, and backups.

This frees users (mainly developers) to focus on providing fast performance, high availability, security and compatibility.

Amazon RDS is available on several database instance types - optimized for memory, performance or I/O - and provides you with six familiar database engines to choose from including Amazon Aurora, MySQL, PostgreSQL, MariaDB, Oracle Database, and SQL Server.



Why we love RDS Databases?

Easy Scalability

High Availability and
managing large amounts
of data.

Automated Backups
and Snapshots

Fully Managed Service



Read Replicas for
Improved Performance

Easy to implement

Relational Data Storage

Integration with other AWS
Services

What's up with PostgreSQL?

The database we will talk about today is PostgreSQL and the reason behind that is the experience we acquired when we had two different projects requesting major version upgrades in this engine.

The first of them was not planned at all, hence important factors were not considered in the analysis. For the second, the correct analysis and planning were carried out. Let's analyze::



Unplanned upgrade:

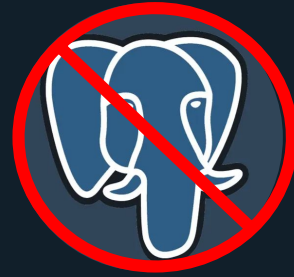
- API version was not compatible with the new version of PostgreSQL.
- Errors in deprecated data types, reg types and extensions.
- The DB had to be restored to its previous version.
- The restore caused slowness in reads to the DB.
- DB restore and analyze caused an 18 hour delay in API transactions.

Planned upgrade:

- API version was compatible with the PostgreSQL version.
- Fixed deprecated data types and corresponding extensions.
- A restore was not necessary and the impact of slowness was minimized with the appropriate analyze and reindex.
- The upgrade took one hour in total to complete. There was no delay in transactions.

Don'ts of a safe Upgrade

It is important to recognize the practices that can put our upgrade at risk. Taking the time for a detailed analysis can save us from having a lot of headaches. **Avoid doing this:**



- Not doing prior research. Versions, End of Life support dates, compatibility and deprecated features.
- Not reviewing parameters and making the necessary adjustments.
- Not planning a Backup and Restore strategy.
- Not having a contingency plan in case of slowness.
- Skip validation of data, sessions and memory usage.

Do's of a safe Upgrade

Phases to plan an Upgrade in PostgreSQL RDS



Before The Upgrade:

1. Checking end of life versions and suitable Major Version
2. Checking Matrix of features
3. Testing the Major Version Upgrade
4. Validating Parameter groups and Reg Data Types

After The Upgrade:

1. Analyze and Reindex
2. Restore back in case of failure
3. Interesting hacks to avoid slowness
4. Delete Test Endpoints

During The Upgrade:

1. Performing a secure Backup
2. Major Version Upgrade with AWS Console

Before The Upgrade

1. Checking end of life versions and suitable Major Version

Before deciding which will be the next version that will best suit the needs for our clusters, we must consider the dates when versions will be no longer supported.

You can review the full list for end of life of each version in this [AWS Docs link](#) in the section named: "Release calendar for Amazon RDS for PostgreSQL".

Refer the following Link for more information:

<https://docs.aws.amazon.com/AmazonRDS/latest/PostgreSQLReleaseNotes/postgresql-release-calendar.html>

Release calendar for Amazon RDS for PostgreSQL

RDS for PostgreSQL major versions remain available at least until community end of life for the corresponding community version. You can use the following dates to plan your testing and upgrade cycles. These dates represent the earliest date that an upgrade to a newer version might be required. If Amazon RDS extends support for an RDS for PostgreSQL version for longer than originally stated, we plan to update this table to reflect the later date.

Note

Dates with only a month and a year are approximate and are updated with an exact date when it's known.

PostgreSQL major version	Community release date	RDS release date	Community end of life date	RDS end of standard support date
PostgreSQL 15 Current minor version: 15.3	09 February 2023	27 February 2023	November 2027	November 2027
PostgreSQL 14 Current minor version: 14.8	30 September 2021	3 February 2022	12 November 2026	November 2026
PostgreSQL 13 Current minor version: 13.11	24 September 2020	24 February 2021	November 2025	November 2025
PostgreSQL 12 Current minor version: 12.15	3 October 2019	31 March 2020	14 November 2024	November 2024
PostgreSQL 11 Current minor version: 11.20	18 October 2018	13 March 2019	9 November 2023	November 2023
PostgreSQL 10 Current minor version:	5 October 2017	27 February 2018	10 November 2022	April 2023

Before The Upgrade

2. Checking Matrix of features

It is possible that when doing a Major Version Upgrade some functionalities are no longer compatible between versions, to reduce the impact of this, when choosing the new version we must consider the features matrix for PostgreSQL. In the following example we can observe the differences in data types, functions and operators in the different versions starting from version 11.

The complete list of comparisons is available in this link: <https://www.postgresql.org/about/featurematrix/>

Data Types, Functions, & Operators	16	15	14	13	12	11
Arrays of compound types	Yes	Yes	Yes	Yes	Yes	Yes
Array support	Yes	Yes	Yes	Yes	Yes	Yes
ENUM data type	Yes	Yes	Yes	Yes	Yes	Yes
GUID/UUID data type	Yes	Yes	Yes	Yes	Yes	Yes
macaddr8 data type	Yes	Yes	Yes	Yes	Yes	Yes
Multiranges	https://www.postgresql.org/docs/10/static/datatype-net-types.html#datatype-macaddr8	Yes	Yes	No	No	No
NULLs in Array	Yes	Yes	Yes	Yes	Yes	Yes
Phrase search	Yes	Yes	Yes	Yes	Yes	Yes
Range types	Yes	Yes	Yes	Yes	Yes	Yes
smallserial type	Yes	Yes	Yes	Yes	Yes	Yes
Type modifier support	Yes	Yes	Yes	Yes	Yes	Yes
XML data type	Yes	Yes	Yes	Yes	Yes	Yes

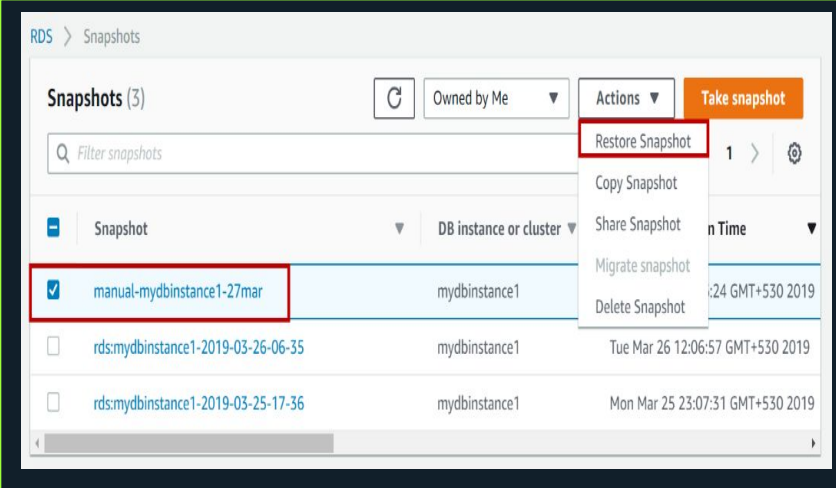
Before The Upgrade

3. Testing the Major Version Upgrade

Before upgrading PostgreSQL DB clusters to a new major version, we strongly recommend that you test the upgrade to verify that all your applications work correctly.

Making a copy of the original DB is the best alternative before starting any upgrade process, this will allow you to make the corresponding tests on the application side without affecting the original DB. Furthermore, it could be easily rebuilt in case of detecting any issue, discarding this way the need of a regression plan.

Note: Keep in mind that a Major Version Upgrade implies a period of downtime, which will leave your applications without service, it is important to plan well the maintenance window. The time that the upgrade can take is relative, an estimate can be obtained during the testing on a copy of our DB.



The screenshot displays the AWS RDS Snapshots console. At the top, there is a breadcrumb 'RDS > Snapshots', a refresh button, a filter 'Owned by Me', and an 'Actions' dropdown menu. A 'Take snapshot' button is also visible. Below this is a search bar 'Filter snapshots' and a table of snapshots. The first row is selected, and its 'Actions' menu is open, showing options like 'Restore Snapshot', 'Copy Snapshot', 'Share Snapshot', 'Migrate snapshot', and 'Delete Snapshot'. The table has columns for 'Snapshot', 'DB instance or cluster', and 'Created Time'.

Snapshot	DB instance or cluster	Created Time
<input checked="" type="checkbox"/> manual-mydbinstance1-27mar	mydbinstance1	Mon Mar 27 23:07:31 GMT+530 2019
<input type="checkbox"/> rds:mydbinstance1-2019-03-26-06-35	mydbinstance1	Tue Mar 26 12:06:57 GMT+530 2019
<input type="checkbox"/> rds:mydbinstance1-2019-03-25-17-36	mydbinstance1	Mon Mar 25 23:07:31 GMT+530 2019

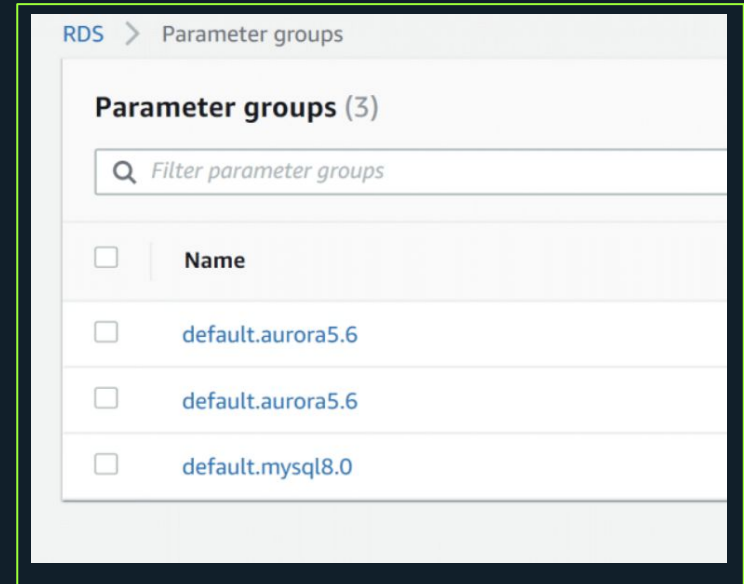
Before The Upgrade

4. Validating Parameter groups and Reg Data Types

Getting Parameter Groups is really important to avoid that parameters remain different between versions (in case we are using the Custom ones). If the DB is using Default parameter groups, the upgrade process will automatically assign the one corresponding to the new version, on the other hand, if they are of Custom type, we will have to follow the recommendations of the AWS Docs:

1. Specify the default DB instance, DB cluster parameter group, or both for the new DB engine version.
2. Create your own custom parameter group for the new DB engine version.

Note: If you associate a new DB instance or DB cluster parameter group as a part of the upgrade request, make sure to reboot the database after the upgrade completes to apply the parameters.



Before The Upgrade

4. Validating Parameter groups and Reg Data Types

There are two aspects to consider in preparation for the upgrade, either for testing or the final upgrade. AWS docs recommend verify the Prepared Transactions, Reg Data Types and Templates as follows:

1. Commit or roll back all open prepared transactions before attempting an upgrade. Remove all uses of the reg* data types before attempting an upgrade.
2. To verify that there are no uses of unsupported reg* data types, use the following query for each database.

```
SELECT count(*)
FROM pg_catalog.pg_class c, pg_catalog.pg_namespace
n, pg_catalog.pg_attribute a
  WHERE c.oid = a.attrelid
        AND NOT a.attisdropped
        AND a.atttypid IN
('pg_catalog.regproc'::pg_catalog.regtype,
'pg_catalog.regprocedure'::pg_catalog.regtype,
'pg_catalog.regoper'::pg_catalog.regtype,
'pg_catalog.regoperator'::pg_catalog.regtype,
'pg_catalog.regconfig'::pg_catalog.regtype,
'pg_catalog.regdictionary'::pg_catalog.regtype)
  AND c.relnamespace = n.oid
  AND n.nspname NOT IN ('pg_catalog',
'information_schema');
```

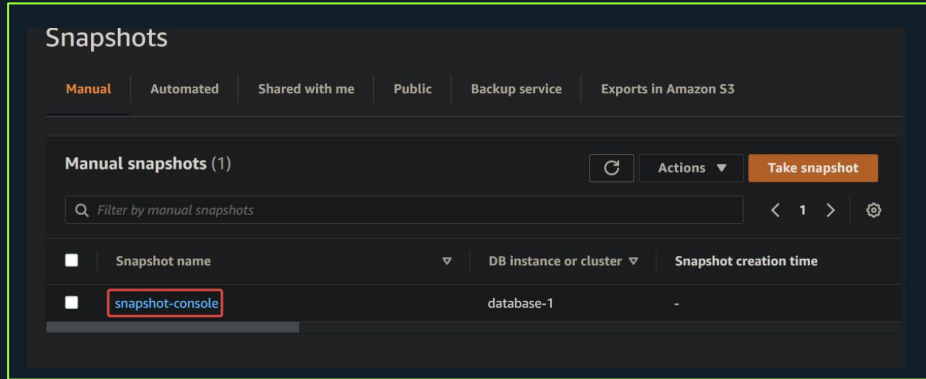
During The Upgrade

1. Performing a secure Backup

The upgrade process creates a DB cluster snapshot of your DB cluster during upgrading, so you can skip this step, although an additional snapshot can give us more security in case we have to revert the Upgrade.

If you also want to do a manual backup before the upgrade process, refer the following link for more information:

<https://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/aurora-restore-snapshot.html>



During The Upgrade

2. Major Version Upgrade with AWS Console

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the navigation pane, choose Databases, and then choose the DB cluster that you want to upgrade.
3. Choose Modify. The Modify DB cluster page appears.
4. For Engine version, choose the new version.
5. Choose Continue and check the summary of modifications.
6. To apply the changes immediately, choose Apply immediately. Choosing this option can cause an outage in some cases.
7. On the confirmation page, review your changes. If they are correct, choose Modify Cluster to save your changes or choose Back to edit your changes or Cancel to cancel your changes.



After The Upgrade

1. Analyze and Reindex

After you complete a major version upgrade, it is important to refresh the statistics with `ANALYZE` and `REINDEX` as follows:

1. Run the `ANALYZE` operation to refresh the `pg_statistic` table.

2. Run `REINDEX` on any hash indexes you have. Hash indexes could change in version upgrade and must be rebuilt. To locate invalid hash indexes, run the following SQL for each database that contains hash indexes:

```
ANALYZE [ ( option [, ...] ) ] [ table_and_columns [, ...] ]  
ANALYZE [ VERBOSE ] [ table_and_columns [, ...] ]
```

```
SELECT idx.indrelid::regclass AS table_name,  
       idx.indexrelid::regclass AS index_name  
FROM pg_catalog.pg_index idx  
     JOIN pg_catalog.pg_class cls ON cls.oid =  
       idx.indexrelid  
     JOIN pg_catalog.pg_am am ON am.oid = cls.relam  
WHERE am.amname = 'hash'  
AND NOT idx.indisvalid;
```

```
REINDEX [ ( option [, ...] ) ] { INDEX | TABLE |  
SCHEMA | DATABASE | SYSTEM } [ CONCURRENTLY ] name
```

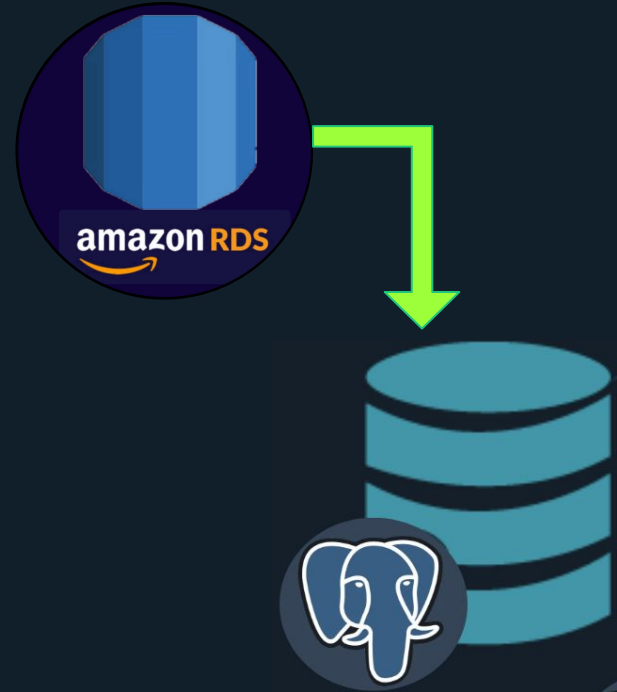

After The Upgrade

2. Restore back in case of failure

When you initiate the upgrade process to a new major version, Aurora PostgreSQL takes a snapshot of the Aurora DB cluster before it makes any changes to your cluster. This snapshot is created for major version upgrades only, not minor version upgrades. The snapshot name includes preupgrade as its prefix, the name of your Aurora PostgreSQL DB cluster, the source version, the target version, and the date and timestamp:

`Preupgrade-endpoint-rds-global-db-12-8-to-14-06-2024-01-13-00-21`

Besides this preupgrade snapshot, you can use the backup snapshot that you take in the Perform a Backup to restore back your cluster to the previous version before the Upgrade.



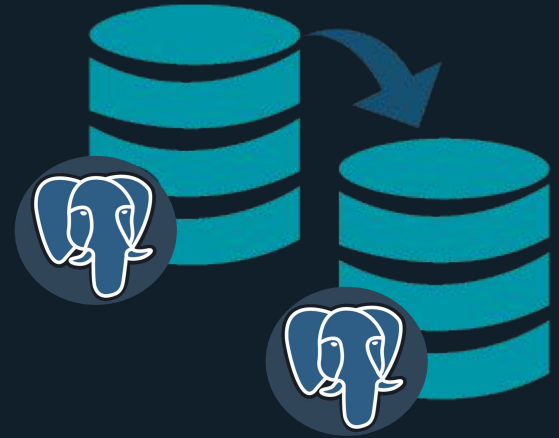
After The Upgrade

3. Interesting hacks to avoid slowness

A RDS database that was recently restored from a snapshot will be slower than a usual database. In a restored database, a query that would normally take up to 1-2 seconds can run for more than a minute and even time out.

Why?

The reason is that RDS snapshots are stored in S3 and when you restore a database from a snapshot AWS doesn't copy the data from S3 to the RDS DB's underlying EBS volume. Instead, the data stays in S3 until the database tries to access it. Only when you run a query for the first time does the queried data get copied to the RDS DB. This lazy loading is what causes the slowness.



After The Upgrade

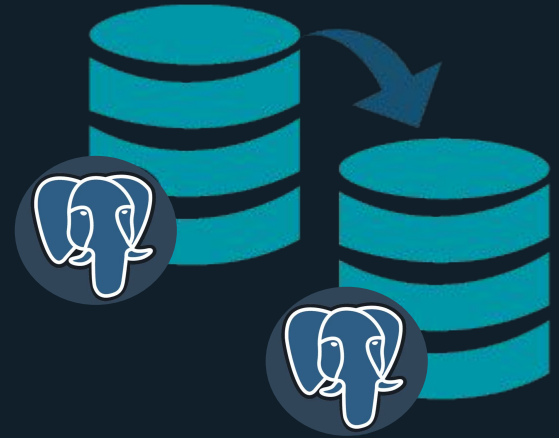
3. Interesting hacks to avoid slowness

What to do?

We can let the lazy loading work, and sometime, eventually the data will get copied over. However, that is often not acceptable.

One way to optimize the reads and force the optimizer to draw explain plans, we can run "Select * From Table" queries to each table, but the beautiful PostgreSQL offers us the option of running a **vacuum analyze**, which will update the plan full table scan of all objects.

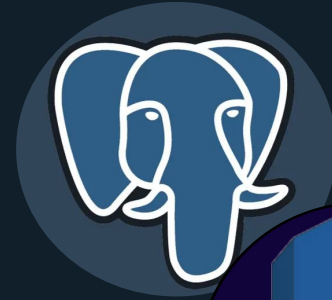
For smaller databases (less than 100Gb) this won't take much time. However, for bigger ones, a vacuum may take many hours. Once it finishes, the slowness and delays disappear. It's important to consider this in our planning before performing any upgrade.



Summary

Don't forget:

- Make thorough planning **before** any action.
- Stay aware of End of Life dates to avoid hasty actions
- Invest all the necessary time in testing and time estimates, downtime is a very important factor.
- Be sure to decommission all unnecessary endpoints and snapshots to avoid excessive charges on your bill.
- If you have any questions, **call your DBA trusted friend!**
- **Enjoy** your Database in its new version!





**¡Gracias!
Keep on Rocking!**



@joyce.dbatech



**linkedin.com/in/joselyne
-lopez-0710ab82**

