CRQ000001930210: AWS Aurora RDS PostgreSQL upgrade from 15.x to 16.x

⚠ This document is specific to Aurora RDS PostgreSQL upgrade from 15.x to 16.x. For any other versions do refer to the official AWS documents for those versions and create a separate document for the implementation.

ℹ **Announcement: Amazon Aurora PostgreSQL 12.14 end of support is March 31, 2024**

https://docs.aws.amazon.com/AmazonRDS/latest/PostgreSQLReleaseNotes/postgresql-release-calendar.html#Release.Calendar

ℹ **REFERENCE**

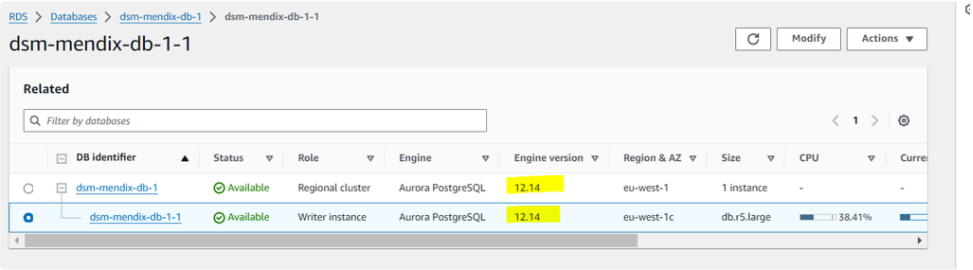
https://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/USER_UpgradeDBInstance.PostgreSQL.html#USER_UpgradeDBInstance.PostgreSQL.UpgradeVersion

Overview of upgrading PostgreSQL

Check current version

Sign in to the AWS Management Console, use the region selector in the navigation bar to choose the AWS Region for your deployment, and open the AWS RDS console at https://console.aws.amazon.com/rds.

In the navigation pane, click **Databases**. From there, select your database instance



Determining which engine version to upgrade to

As per AWS doc,

```
aws --region eu-west-1 rds describe-db-engine-versions  --engine aurora-postgresql  --engine-version 12.14 --query 'DBEngineVersions[].ValidUpgradeTarget[?IsMajorVersionUpgrade == `true`]'
```

Prerequisites

1. **Mendix Compatibility**

Mendix version 8,9,10 supports PostgreSQL 12, 13, 14, 15, 16

2. **Validate Supported DB engines for DB instance classes**

Refer doc

For Aurora Postgresql 16.6, instance class has to be at least db.r5.large

3. **Verify that there are no uses of unsupported *reg** data types, use the following query for each database**

Remove all uses of the *reg** data types before attempting an upgrade.

Except for `regtype` and `regclass`, you can't upgrade the *reg** data types. The pg_upgrade utility can't persist this data type, which is used by Amazon Aurora to do the upgrade.

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');
```

4. **To find the `unknown` data type in your database:**

```
SELECT DISTINCT data_type FROM information_schema.columns WHERE data_type ILIKE 'unknown';
```

5. **To list your currently installed extension**

```
SELECT * FROM pg_extension;
```

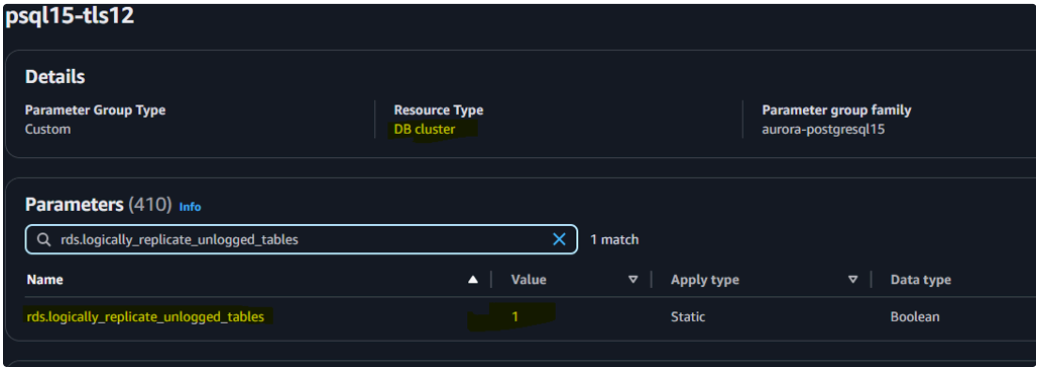
6. **To locate invalid hash indexes, run the following SQL for each database that contains hash indexes.**

```
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;
```

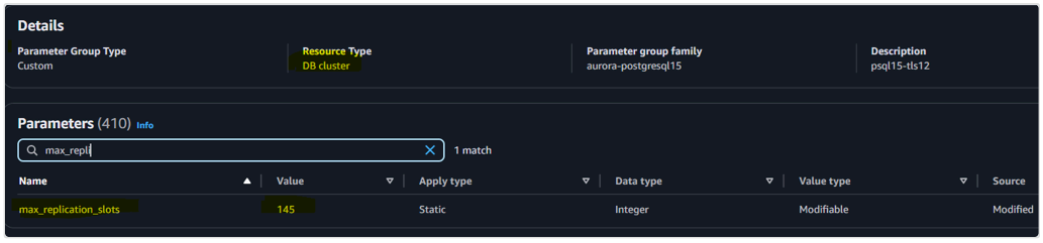
Update the below DB cluster parameters in blue Database

1. **rds.logically_replicate_unlogged_tables** should be on (set value as 1).

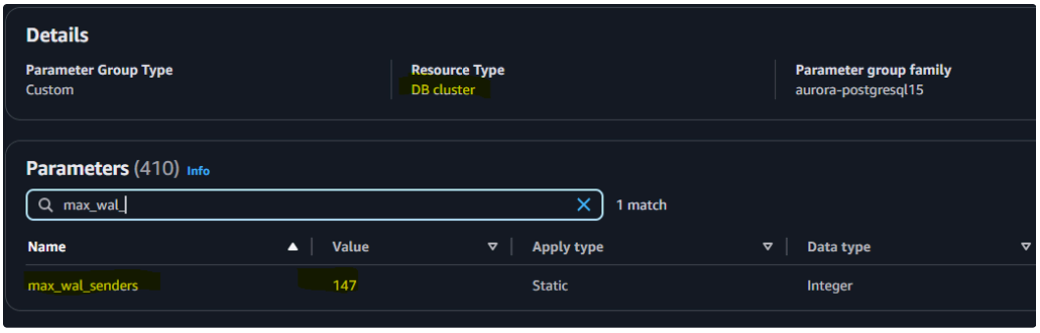


2. **rds.logical_replication** should be on (set value as 1)

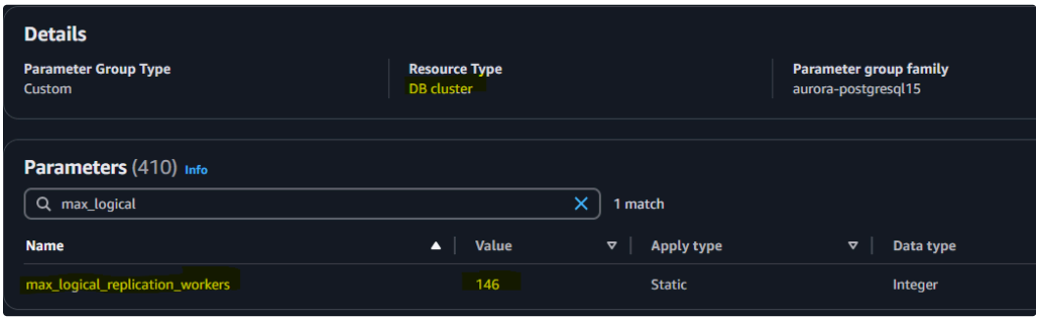
3. **max_replication_slots** – This must be set to at least the number of subscriptions expected to connect, plus some reserve for table synchronization. There will be one subscription per database, so make sure to set a number greater than the number of databases.



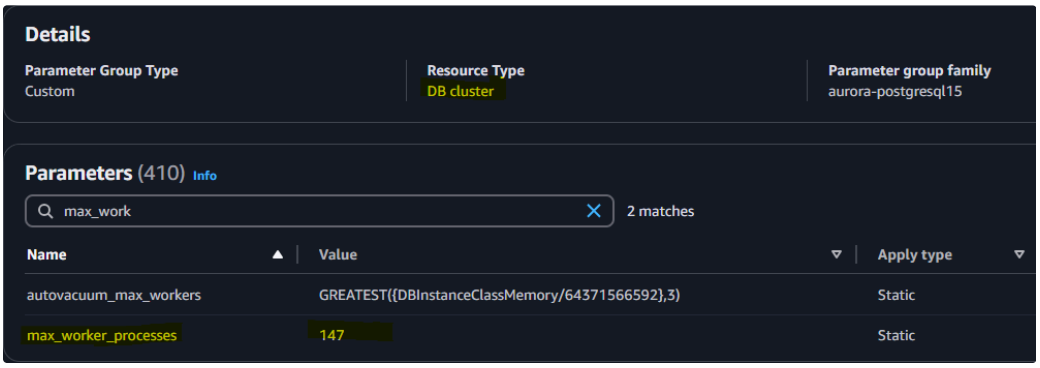
4. **max_wal_senders** – This is the maximum number of background processes that the system can support. It's recommended to set this number slightly higher than max_replication_slots.



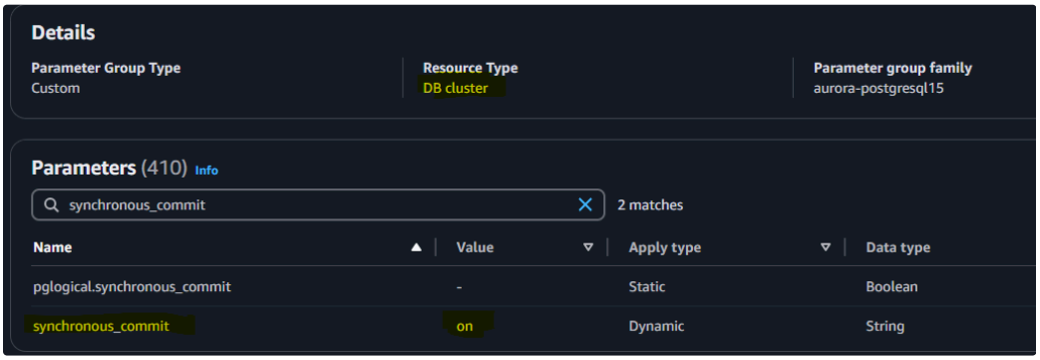
5. **max_logical_replication_worker** – You should set this to a number of databases, plus some reserve for the table synchronization workers and parallel apply workers.



6. **max_worker_processes** – This is the maximum number of background processes that the system can support. It should be set to at minimum max_logical_replication_worker + 1 or higher.



7. In addition, make sure that the **'synchronous_commit'** parameter is set to on.



- **After you configure the required parameters, reboot the DB instance so that your changes take effect.**

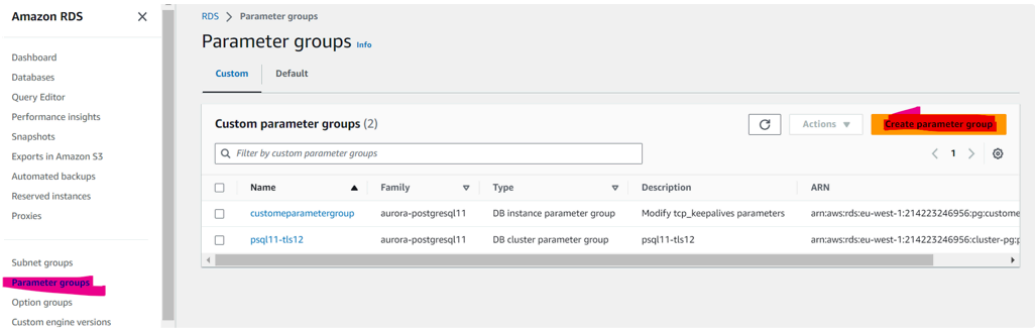
Note : Make sure that all tables in the DB cluster have a primary key. PostgreSQL logical replication doesn't allow UPDATE or DELETE operations on tables that don't have a primary key.

- **Create cluster parameters for higher (selected) version of AWS RDS (16.6 V).**

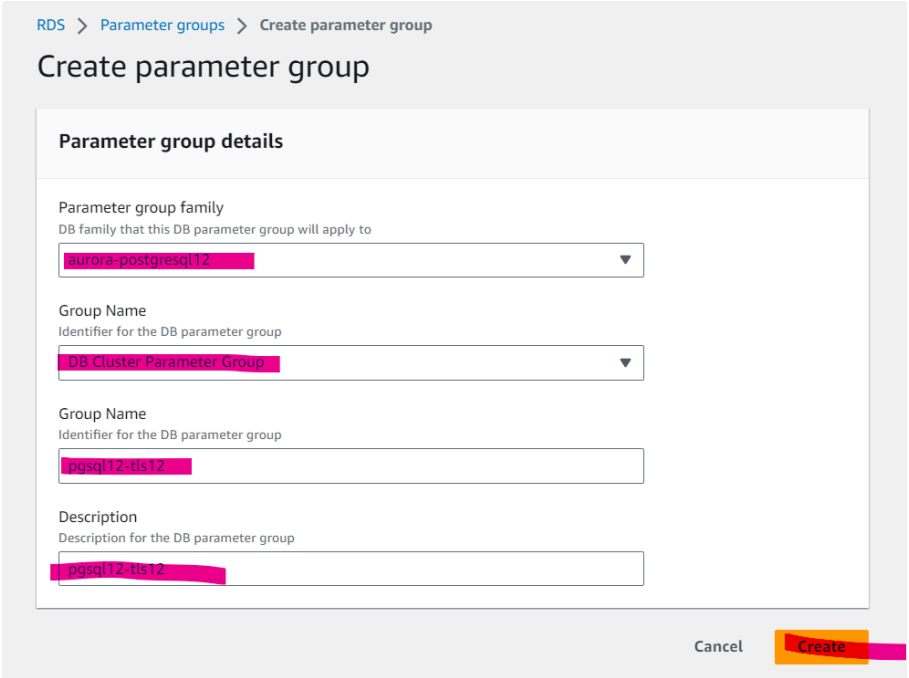
As the Mendix application will only support the TLS 1.2 version, we need to create a custom parameter group for the database (v16.6) upgrade for both the DB instance and DB cluster. In the parameter group, we should set the TLS v1.2 as the default value.

To create custom parameter group and set the TLS v1.2 as the default value for the both the DB instance and DB cluster by using the console

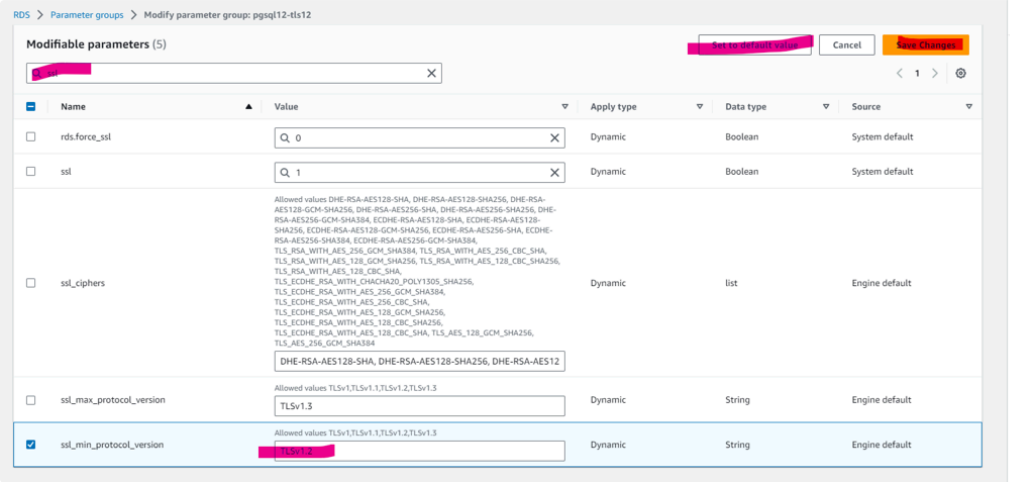
1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose **Parameter groups**, and then click on the create parameter group



3. On the "**Create parameter group**" page, select the desired parameter group family, specify the Group Name (either DB parameter or DB cluster parameter), provide a Group Name and Description, and then click on the "Create" button.



4. To set the TLS 1.2 version as the default, open the custom created parameter group, click on "Edit," search for "ssl_min_protocol_version," which contains the TLS 1.2 version, then click on "Set to default value," and finally, save the changes.



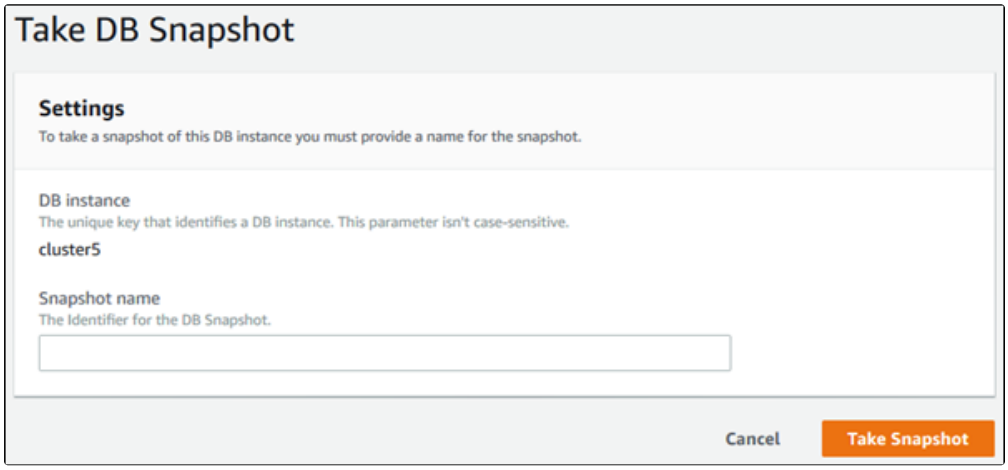
- **Create a Manual Snapshot of AWS RDS**

The upgrade process creates a DB cluster snapshot of your DB cluster during upgrading. If you also want to do a manual backup before the upgrade process, follow below steps:

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**.
3. In the list of DB instances, choose a writer instance for the DB cluster.
4. Choose **Actions**, and then choose **Take snapshot**.

The **Take DB Snapshot** window appears.

5. Enter the name of the DB cluster snapshot in the **Snapshot name** box.



6. Choose **Take Snapshot**.

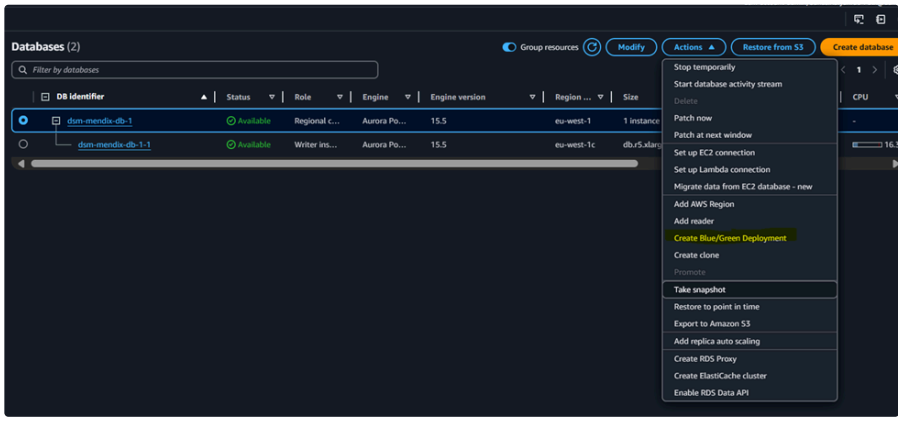
- **Check for unsupported usage:**

Commit or roll back all open prepared transactions before attempting an upgrade. You can use the following query to verify that there are no open prepared transactions on your instance.

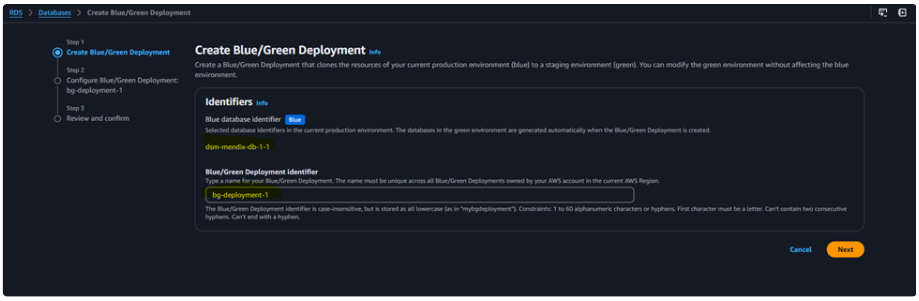
```
SELECT count(*) FROM pg_catalog.pg_prepared_xacts;
```

Steps to create blue/green Deployment

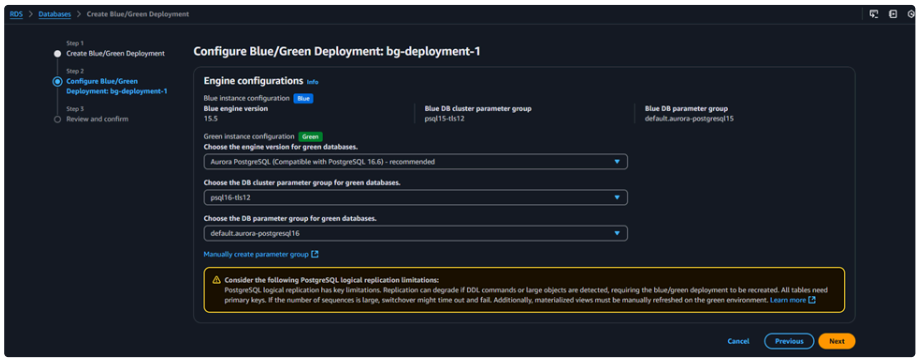
1. Navigate to AWS PostgreSQL RDS
2. Select DB cluster and click on Action then choose Blue/Green Deployment



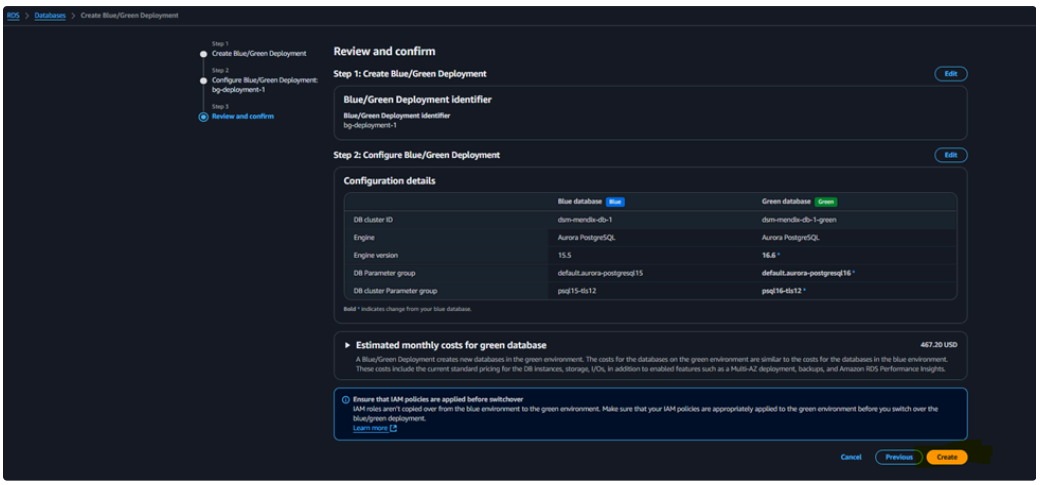
3. Review the blue database identifiers. Make sure that they match the DB instances that you expect in the blue environment. If they don't, choosse cancel.
4. For Blue/Green Deployment identifier, enter a name for your blue/green deployment.



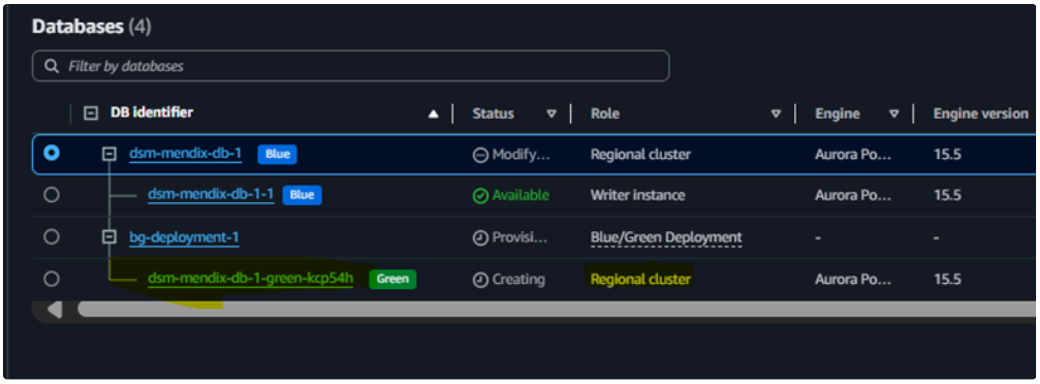
5. Select higher engine version of Green deployment
6. Select the parameter group.



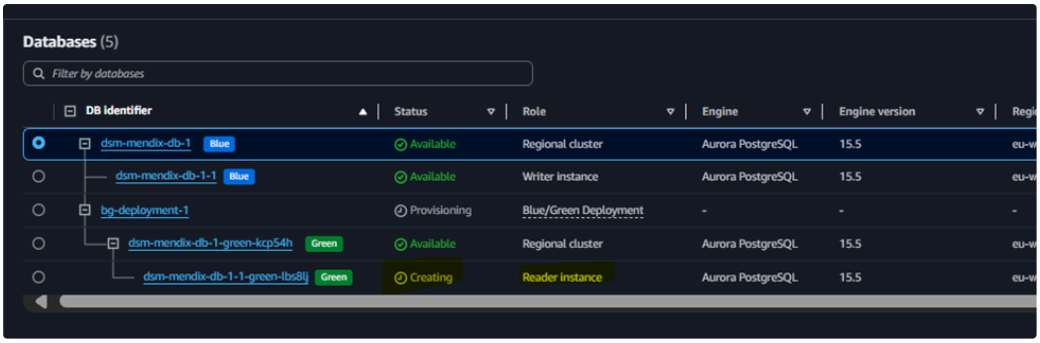
7. [In the remaining sections, specify the settings for the green environment. For information about each setting, see Settings for creating blue/green deployments.]
8. Choose Create staging environment.



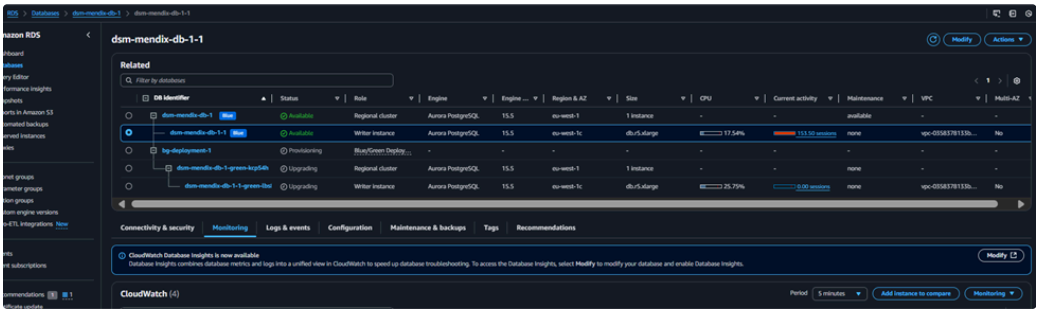
Cluster Creating Status



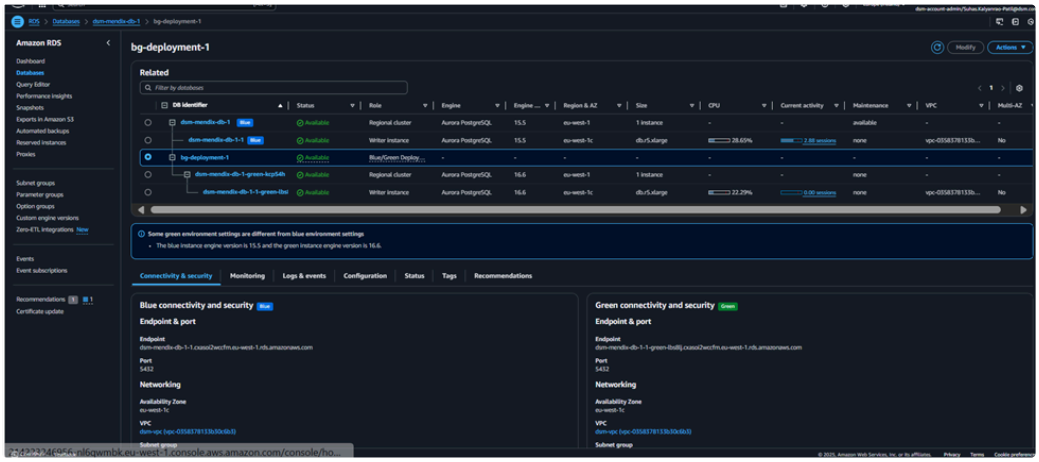
DB instance creating status



DB upgrade status



Available status of blue/green deployment



9. [For information about modifying a DB cluster, see Modifying an Amazon Aurora DB cluster.]

Test your staging environment.

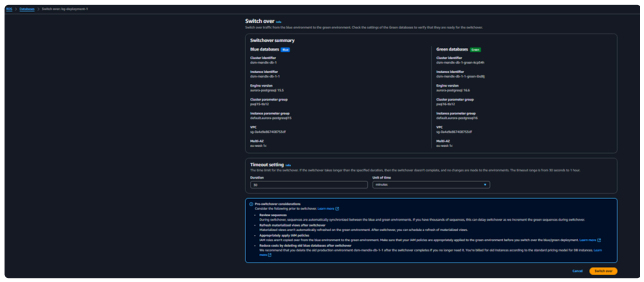
During testing, we recommend that you keep your databases in the green environment read only. Enable write operations on the green environment with caution because they can result in replication conflicts. They can also result in unintended data in the production databases after switchover. For Aurora PostgreSQL, set the default_transaction_read_only parameter to off at the session level

Validate db connections

- Connect to pgadmin and connect to the few application database and check
- Validate application logs
- Validate from the RDS monitoring graphs that sessions are coming in to the database

Switching over a blue/green deployment

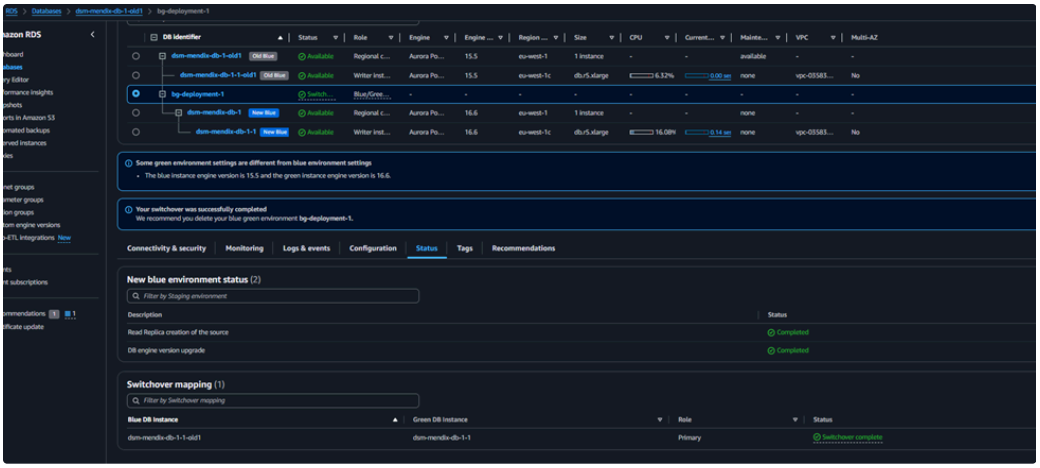
1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/.
2. In the navigation pane, choose Databases, and then choose the blue/green deployment that you want to switch over.
3. For Actions, choose Switch over.
4. On the Switch over page, review the switchover summary. Make sure the resources in both environments match what you expect. If they don't, choose Cancel.



5. For Timeout settings, enter the time limit for switchover.

Deleting a blue/green deployment in Amazon Aurora

1. Sign in to the AWS Management Console and open the Amazon RDS console at https://console.aws.amazon.com/rds/
2. Before delete validate the status of switchover mapping as it should be switchover complete as shown in below snap.



3. In the navigation pane, choose Databases, and then choose the blue/green deployment that you want to delete.
4. For Actions, choose Delete. - The Delete Blue/Green Deployment? window appears.
5. To delete the green databases, select Delete the green databases in this Blue/Green Deployment.
6. Enter delete me in the box.
7. Choose Delete.

Upgrade certain extensions to the latest available version before performing the major version upgrade. The extensions to update include the following:

- `pgRouting`
- `postgis_raster`
- `postgis_tiger_geocoder`
- `postgis_topology`
- `address_standardizer`
- `address_standardizer_data_us`

Run the following command for each extension that you are using.

```
ALTER EXTENSION PostgreSQL-extension
UPDATE TO 'new-version'
```

Drop `unknown` data types, depending on your target version.

PostgreSQL version 10 stopped supporting the `unknown` data type. If a above 10

To find the `unknown` data type in your database so you can remove the offending column or change it to a

version database uses the `unknown` data type, an upgrade to a version 10 shows an error message such as the following.

```
Database instance is in a state that cannot be upgraded: PreUpgrade checks failed:
The instance could not be upgraded because the 'unknown' data type is used in user tables.
Please remove all usages of the 'unknown' data type and try again."
```

supported data type, use the following SQL code.

```
SELECT DISTINCT data_type FROM information_schema.columns WHERE data_type ILIKE 'unknown';
```

- **Upgrade PostgreSQL extensions (If applicable)**

A PostgreSQL engine upgrade doesn't automatically upgrade any PostgreSQL extensions. To update an extension after an engine upgrade, use the `ALTER EXTENSION UPDATE` command.

> If you are running the `PostGIS` extension in your Amazon RDS PostgreSQL DB instance, make sure that you follow the PostGIS upgrade instructions in the PostGIS documentation before you upgrade the extension.

To upgrade an extension, use the following command.

```
ALTER EXTENSION extension_name UPDATE TO 'new_version';
```

To list your currently installed extensions, use the PostgreSQL pg_extension catalog in the following command.

```
SELECT * FROM pg_extension;
```

To view a list of the specific extension versions that are available for your installation, use the PostgreSQL pg_available_extension_versions view in the following command.

```
SELECT * FROM pg_available_extension_versions;
```

- **After you complete a major version upgrade, following is recommended. At least for large databases execute ANALYZE.**

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

Connect to pgadmin and connect to the each database and execute:

```
ANALYZE VERBOSE;
```

Run `REINDEX` on any hash indexes

If you upgraded to PostgreSQL version 16, run `REINDEX` on any hash indexes you have. Hash indexes were changed in version 16 and must be rebuilt. To locate invalid hash indexes, run the following SQL for each database that contains hash indexes.

```
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;
```