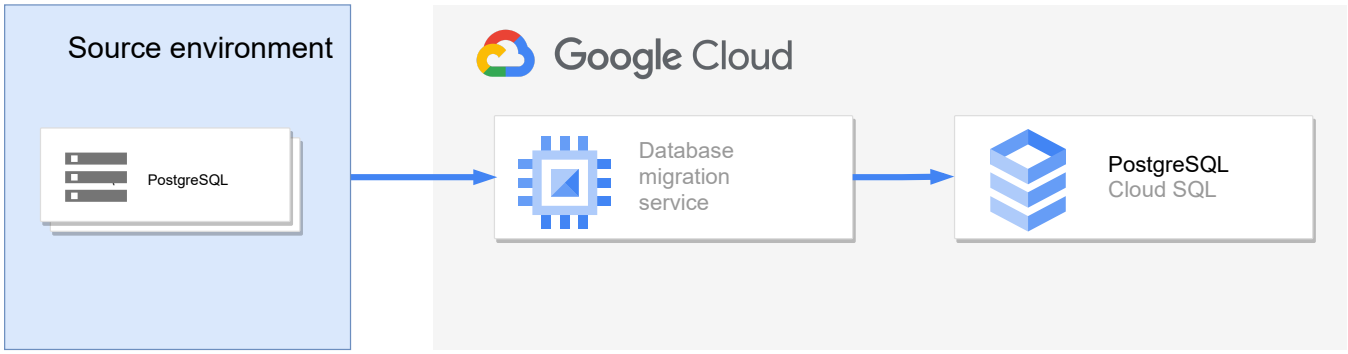
PostgreSQL Migration with Database Migration Service

This document describes how to migrate a PostgreSQL instance and its databases using the Database Migration Service (/database-migration). It outlines various preparation steps and best practices for the whole migration process, including caveats and issues. This document is for data architects and engineers responsible for migrating PostgreSQL to Cloud SQL for PostgreSQL. The document also includes instructions for accomplishing a full migration of PostgreSQL to Cloud SQL for PostgreSQL.

Managing homogeneous database migration

The Database Migration Service for PostgreSQL (/database-migration/docs/postgres) implements the homogeneous database migration of all databases in a source PostgreSQL instance into a Cloud SQL for PostgreSQL instance. That Cloud SQL for PostgreSQL instance is purposely created as the target instance. The following diagram shows the flow of information:



Database Migration Service is a managed Google Cloud service. It provides a graphical user interface to set up and start the database migration process as well as to cut over to the target instance. After the migration is complete, Database Migration Service promotes the target instance to be the new source instance.

Database Migration Service migrates all databases from the source instance to the target instance within the same migration job. This method avoids having to create an individual migration job for each database. Before you migrate, you have to prepare the source instance and the source databases. This document discusses those preparations later.

Database Migration Service supports migration across versions (/database-migration/docs/postgres/cross-version-support) as well.

The target instance, databases, and tables with primary keys are automatically created and migrated by Database Migration Service. Tables without primary keys are not. The Migrating tables without a primary key (#migrating-tables-without-pk) section discusses that topic.

Because Database Migration Service is fully managed, it doesn't require any operational or management oversight. A user can fully focus on defining and running the database migration.

Database migration overview

In general, database migration is a multi-step process. The basic steps are described here. The detailed steps you must take to migrate a database appear later in the document.

1. **Prepare the source instance and databases**
    - Apply settings that allow for a zero downtime migration; Database Migration Service requires these settings.
    - Install `pglogical` (https://www.2ndquadrant.com/en/resources/pglogical/).
2. **Specify the migration job**
    - The migration job specification defines the following items:
        - Source and target instances
        - Network connectivity
        - Additional parameters required to set up a migration
    - The migration job specification also provides a connectivity test to ensure that the Database Migration Service service can reach the source instance.
3. **Plan to migrate non-primary-key tables**
    - Create a migration strategy for each table that doesn't have a primary key before starting the migration job.
4. **Prepare for additional exceptions**

- In addition to non-primary key tables, other objects might require special attention; see Preparing to run the database migration job (#prep-the-db-migration).

5. **Start the migration job**
    - Start the data migration after you complete and successfully test your migration job setup tasks.
6. **Validate migrated data (optional)**
    - After you have migrated the data and quiesced the source instance to prevent further changes, optionally verify that all data is in the target instance and its databases.
7. **Promote the target instance**
    - After the data migration is complete and optionally validated, promote the target instance to the new primary instance.
8. **Migrate applications to new primary instance**
    - Migrate the applications that were originally connected to the source instance to the new primary instance.
9. **Tune the primary instance**
    - To optimize performance, tune the primary instance after the applications begin to use it.
10. **Set up high availability and disaster recovery**
    - Depending on the requirements, think about enabling disaster recovery with cross-region replicas for the new primary instance.

These steps are described in detail throughout this article so that you can fully understand all aspects of the complete database migration process in the context of Database Migration Service for PostgreSQL.

Assumptions and expectations

This section describes the assumptions and expectations necessary to to migrate a PostgreSQL instance and its databases using the instructions in this document.

Database engine version

The following instructions are written for PostgreSQL 13. The source PostgreSQL instance runs on Compute Engine. The target Cloud SQL for PostgreSQL version is also version 13.

If you use earlier versions of PostgreSQL, especially version 9.4 or 9.6, this document doesn't apply completely to your case. See Configure your source (/database-migration/docs/postgres/configure-source-database) for more information about those differences.

Restart the source instance

A basic assumption is that you want to minimize the number of times you restart the source instance. Based on this assumption, the article distinguishes between configuration reloads and instance restarts. If it's possible for you to avoid source instance restarts by delaying them and waiting for an instance restart that you can't avoid, the document indicates it.

Migrate all databases

Because it's impossible for you to only migrate a subset of the databases in a PostgreSQL instance, the article assumes that you migrate all databases in an instance.

If you want to migrate only a subset of the databases from the source instance, remove the databases you don't want before starting the migration. Alternatively, you can migrate all databases and drop the databases you don't want from the target instance once the migration completes.

Each approach has trade offs: Moving databases to different instances before the migration might require changing application configurations and other processes that access the source instance. Dropping databases after migration leaves the source environment unaffected, but affects the cost and timing of the migration.

Migrate tables without primary keys

Tables in databases that don't have a primary key aren't migrated automatically by Database Migration Service. You must manually migrate them. The Migrating tables without a primary key (#migrating-tables-without-pk) section outlines different strategies.

Extensions, large objects, and external wrappers

Check whether all the features that you use in your source databases are available in Cloud SQL for PostgreSQL. For all data-type and data-storage approaches you use (like external wrappers), check if `pglogical` can migrate them or if you have to manually migrate them. For more information, see Preparing to run the database migration job (#prep-the-db-migration).

Costs

This tutorial uses the following billable components of Google Cloud:

- Compute Engine (/compute/all-pricing)
- Cloud SQL for PostgreSQL (/sql/pricing#pg-pricing)

To generate a cost estimate based on your projected usage, use the pricing calculator (/products/calculator).

Before you begin

1. In the Google Cloud Console, on the project selector page, select or create a Google Cloud project.

★ **Note**: If you don't plan to keep the resources that you create in this procedure, create a project instead of selecting an existing project. After you finish these steps, you can delete the project, removing all resources associated with the project.

Go to project selector (https://console.cloud.google.com/projectselector2/home/dashboard)

2. Make sure that billing is enabled for your Cloud project. Learn how to confirm that billing is enabled for your project (/billing/docs/how-to/modify-project).

Preparing a database migration

The following instructions show you how to migrate two databases from one source instance. The databases contain regular tables with primary keys, and tables without primary keys.

The instructions help create a source PostgreSQL instance to go through a full database migration with Database Migration Service.

In a production environment, you should already have a source instance that provides a specific throughput and latency to the clients of the various databases.

Source instance access

As you prepare for the migration, ensure that you have sufficient access privileges. You need the appropriate privileges to perform all necessary changes to the source instance and source databases.

Examples of changes that you might have to make appear throughout this document.

Target instance configuration

The database migration job specification asks for configuration information for the target Cloud SQL instance. Providing that information is a best practice. The configuration information helps ensure that your application performs as expected on a Cloud SQL for PostgreSQL instance.

You can best ensure that the instance meets performance requirements by testing the application on a Cloud SQL instance that's configured to meet your throughput and latency requirements. After you have finished your testing and determined the needed Cloud SQL configuration, note down all configuration settings. You need them when specifying a database migration job (#specifying-a-database-migration-job).

If you don't initially test the application, an alternative approach is to configure the target instance like the source instance when creating the migration job. After the cutover completes and the application is accessing the target instance's databases, you can tune the databases. Be aware, however, that this tuning step could possibly occur during production.

Network connectivity

Database Migration Service supports different types of network connectivity (/database-migration/docs/postgres/configure-connectivity). The target instance is a replica of the

source instance and therefore it must be able to connect.

Ensure that your environment supports one of the connectivity types so that you can configure those connectivity types during the migration job specification.

Database schema changes

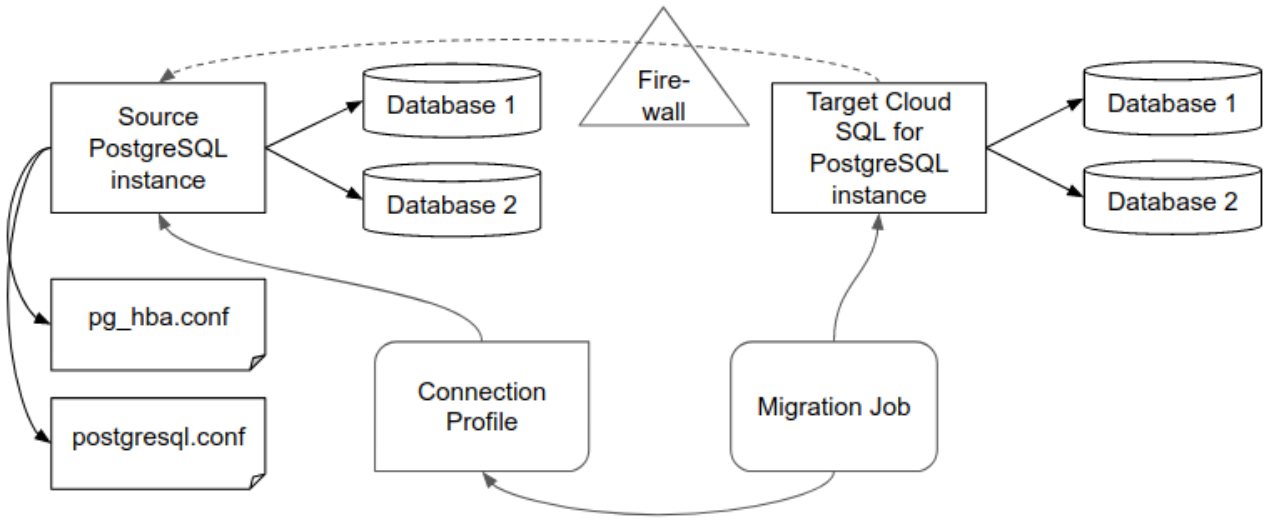
There are three types of schema changes (https://www.postgresql.org/docs/current/ddl.html):

- **Changing an existing schema**: For example, users can change an existing table by adding a column.
- **Adding schema elements**: For example, users can add a new table schema.
- **Removing schema elements**: For example, users can drop an existing table schema.

Determine if any of these types of changes can happen during database migration. If they can, use the information in the Manage schema changes (#manage_schema_changes) section to make the changes.

Overview of artifacts for database migration

The following diagram shows the major database migration artifacts and how they are interrelated. It doesn't show the database tables of the various databases:



The source instance in the preceding diagram consists of two databases. It also contains two configuration files, `pg_hba.conf` and `postgresql.conf`, that you might have to change to migrate data with Database Migration Service.

A Database Migration Service migration job refers to a connection profile for the source instance. This profile refers to the source instance. A Database Migration Service migration job also refers to a Cloud SQL target instance. The target instance is a replica of the source instance (indicated by the dashed arrow).

The target instance connects to the source instance. The source instance must be accessible by the IP address of the target instance. If a firewall is present, it has to allow the IP address of the target Cloud SQL for PostgreSQL instance to connect to the source instance.

Completing these instructions creates a source instance on a Compute Engine VM. To connect to the source instance, you must open the firewall for the IP address the target instance uses.

To demonstrate how to move tables without primary keys, the following example assumes that not all tables have primary keys and that you manually migrate those kinds of tables.

Creating a source PostgreSQL instance

The following steps create an example source PostgreSQL instance. You can follow the instructions to create a new example instance, or use an existing instance.

1. In Cloud Shell, create a Compute Engine instance:

```
gcloud beta compute instances create pg-source-1 \
    --zone=us-west1-b \
    --machine-type=e2-standard-2 \
    --image=ubuntu-2004-focal-v20210223 \
    --image-project=ubuntu-os-cloud \
    --boot-disk-size=10GB
```

2. Use SSH to connect to the Compute Engine instance.
3. Follow the instructions (https://www.postgresql.org/download/linux/ubuntu/) to download and to install PostgreSQL for Ubuntu.

4. Sign in to the PostgreSQL shell:

```
sudo -u postgres psql
```

5. Query the server version:

```
show server_version;
```

   Write down the major version. Use it when you specify a migration job.

6. List the databases and observe that the standard databases are present:

```
\l
```

   The output is similar to the following list of databases:

```
                              List of databases
   Name    |  Owner   | Encoding | Collate |  Ctype  |   Access privileges
-----------+----------+----------+---------+---------+-----------------------
 postgres  | postgres | UTF8     | C.UTF-8 | C.UTF-8 |
 template0 | postgres | UTF8     | C.UTF-8 | C.UTF-8 | =c/postgres
           |          |          |         |         | postgres=CTc/postgre
 template1 | postgres | UTF8     | C.UTF-8 | C.UTF-8 | =c/postgres
           |          |          |         |         | postgres=CTc/postgre
```

Create sample source databases

1. In the PostgreSQL instance, create two databases:

```
CREATE DATABASE dmspg_1;
CREATE DATABASE dmspg_2;
```

2. Confirm their creation:

```
\l
```

The output is similar to the following list of databases:

```
                              List of databases
   Name    |  Owner   | Encoding | Collate |  Ctype  |   Access privileges
-----------+----------+----------+---------+---------+-----------------------
 dmspg_1   | postgres | UTF8     | C.UTF-8 | C.UTF-8 |
 dmspg_2   | postgres | UTF8     | C.UTF-8 | C.UTF-8 |
 postgres  | postgres | UTF8     | C.UTF-8 | C.UTF-8 |
 template0 | postgres | UTF8     | C.UTF-8 | C.UTF-8 | =c/postgres
           |          |          |         |         | postgres=CTc/postgres
 template1 | postgres | UTF8     | C.UTF-8 | C.UTF-8 | =c/postgres
           |          |          |         |         | postgres=CTc/postgres
(5 rows)
```

3. Connect to the first database and create two tables in each database:

```
\c dmspg_1

CREATE TABLE accounts (
    user_id VARCHAR(128) PRIMARY KEY,
    username VARCHAR (128) NOT NULL);
CREATE TABLE notes (
    note VARCHAR(256));
```

4. Confirm that the system created the database tables:

```
\dt
```

The output is similar to the following list of relations:

```
          List of relations
 Schema |   Name   | Type  |  Owner
--------+----------+-------+----------
```

```
 public | accounts | table | postgres
 public | notes    | table | postgres
(2 rows)
```

5. Insert some rows into the tables:

```
INSERT INTO accounts (user_id, username)
    VALUES('one','Alice');
INSERT INTO accounts (user_id, username)
    VALUES('two','Bob');
INSERT INTO notes (note)
    VALUES('Initial note');
INSERT INTO notes (note)
    VALUES('Second note');
INSERT INTO notes (note)
    VALUES('Second note');
```

   Note that the same row is added twice to the `notes` table because it doesn't have a primary key to prevent duplicates.

6. Confirm that the rows were inserted into the tables:

```
SELECT * FROM accounts;
SELECT * FROM notes;
```

7. Create the same tables and insert the same rows into the second database `dmspg_2`. If you would like to have different tables in the second database, create them.

8. Exit the PostgreSQL shell:

```
exit;
```

At this point, the instance contains two databases with two tables each. One of the tables doesn't have a primary key and two rows have the same value. This lets you verify that the suggested migration approach (#migrating-tables-without-pk) works.

Prepare the source instance and databases

Database Migration Service requires specific preparation steps for the source instance, and for the databases (/database-migration/docs/postgres/configure-source-database). In the following section, preparing the source instance and preparing the source databases are discussed separately.

Prepare the source instance

After configuring the source instance, reload it to apply the new configuration values.

1. In Cloud Shell, use SSH to connect to the Compute Engine instance.
2. Install `pglogical` by following these instructions: Installation Instructions for `pglogical` (https://www.2ndquadrant.com/en/resources/pglogical/pglogical-installation-instructions/).

   The installation step for PostgreSQL 13 on Ubuntu is as follows:

   ```
   sudo apt-get install postgresql-13-pglogical
   ```

3. Log in to the PostgreSQL shell:

   ```
   sudo -u postgres psql
   ```

4. Run the following commands to change the configuration of the instance (the code sample uses the default PostgreSQL values):

   ```
   ALTER SYSTEM SET shared_preload_libraries = 'pglogical';
   ALTER SYSTEM SET wal_level = 'logical';
   ALTER SYSTEM SET max_replication_slots = 10;
   ALTER SYSTEM SET max_wal_senders = 10;
   ALTER SYSTEM SET max_worker_processes = 8;
   ```

   See Configure your source (/database-migration/docs/postgres/configure-source-database#pglogical) for a discussion about deciding on the values that apply to your situation.

5. Reload the instance configuration:

```
SELECT * FROM pg_reload_conf();
```

6. Check if an instance restart is required for one of the configurations:

```
SELECT pending_restart FROM pg_settings WHERE name = 'max_replication_slots
```

   As a best practice, check all the settings that you set. Doing so helps ensure that you only restart the instance if one or more setting changes requires it.

   The output is as follows:

```
pending_restart
-----------------
 t
(1 row)
```

   The output shows that **pending_restart** is true. That means a reload of the configuration is insufficient and an instance restart is required.

7. Exit from **psql**:

```
exit;
```

8. Restart the instance from the **ssh** shell:

```
sudo systemctl restart postgresql@13-main
```

   This is an instance restart. It affects the accessing database clients. You can consider delaying the instance restart at this point until you know that there are no other additional configuration changes that also require an instance restart, for example, connecting to an

instance from an IDE (#connecting-from-ide) or defining a connectivity method (#define_a_connectivity_method).

9. To check that the configuration is correct, log in to `psql` again:

```
sudo -u postgres psql
```

10. Check that the configuration values are set to the values you specified earlier:

```
SHOW shared_preload_libraries;
SHOW wal_level;
SHOW max_replication_slots;
SHOW max_wal_senders;
SHOW max_worker_processes;
```

The output is similar to:

```
shared_preload_libraries
--------------------------
pglogical
(1 row)

postgres=# SHOW wal_level;
wal_level
-----------
logical
(1 row)

postgres=# SHOW max_replication_slots;
max_replication_slots
----------------------
11
(1 row)

postgres=# SHOW max_wal_senders;
max_wal_senders
-----------------
10
(1 row)
```

```
postgres=# SHOW max_worker_processes;
max_worker_processes
---------------------
8
(1 row)
```

11. Exit the PostgreSQL shell:

```
exit;
```

Create a migration user

In order to migrate the source databases, a user must have certain privileges on all user databases. In the following steps, you create a migration user for this purpose. If you already have a user defined in the instance for this purpose, skip these steps and use that user profile.

After the migration completes, this user no longer serves a purpose. You can remove it from the source instance.

1. In Cloud Shell, log in to the PostgreSQL shell:

```
sudo -u postgres psql
```

2. Create the migration user `dbmig`:

```
CREATE USER dbmig WITH ENCRYPTED PASSWORD 'dbmig';
```

3. Set the replication role:

```
ALTER USER dbmig WITH REPLICATION;
```

4. Exit the PostgreSQL shell:

```
exit;
```

You might require the user `postgres` to have a password. The command that sets an example password is `ALTER USER postgres PASSWORD 'postgres'`. In general it's a good practice to set passwords for users, so we highly recommend it. If you require a password login and not a peer login, you must change the `pg_hba.conf` file (located at `/etc/postgresql/13/main/pg_hba.conf`). See the `pg_hba.conf` file (https://www.postgresql.org/docs/current/auth-pg-hba-conf.html), for more information.

If you plan to connect to any database during the tutorial, set the password for `postgres` now so that you don't have to come back to this section.

Prepare the user databases

After the instance is configured, every user database within it requires configuration (/database-migration/docs/postgres/configure-source-database). This includes the two databases that you created earlier (`dmspg_1` and `dmspg_2`) and `postgres`.

1. In Cloud Shell, log in to the PostgreSQL shell:

   ```
   sudo -u postgres psql
   ```

2. List all user databases:

   ```
   \l
   ```

   This step helps to ensure that you can see all the user databases. You start to configure them in the next step.

3. Connect to each `<user_database>` (except for `template0` and `template1`) and run the following commands:

   ```
   \c <user_database>
   CREATE EXTENSION IF NOT EXISTS pglogical;
   ```

4. Determine all user schemas in each user database:

   ```
   \dn
   ```

   If you followed the instructions so far and didn't create schemas, the only schema in each database is `public` and `pglogical`.

5. Run the following commands (in each user database) for each `<schema>` that isn't `pglogical`:

   ```
   GRANT USAGE on SCHEMA <schema> to dbmig;
   GRANT SELECT on ALL TABLES in SCHEMA <schema> to dbmig;
   GRANT SELECT on ALL SEQUENCES in SCHEMA <schema> to dbmig;
   ```

6. Run the following commands for the `pglogical` schema in each user database:

   ```
   GRANT USAGE on SCHEMA pglogical to dbmig;
   GRANT SELECT on ALL TABLES in SCHEMA pglogical to dbmig;
   ```

   The Database Migration Service migration job that you specify later only migrates user database tables that have a primary key. You must manually migrate the tables in each user database that don't have a primary key.

7. In each user database, determine all tables that don't have a primary key.

   a. List all databases:

      ```
      \l
      ```

   b. The user databases are `dmspg_1`, `dmspg_2`, and `postgres`. For each, run the following query as outlined in Debugging and other tools (/database-migration/docs/postgres/debugging-tools#find_tables_without_primary_keys_pks):

```
select tab.table_schema,
       tab.table_name
from information_schema.tables tab
left join information_schema.table_constraints tco
          on tab.table_schema = tco.table_schema
          and tab.table_name = tco.table_name
          and tco.constraint_type = 'PRIMARY KEY'
where tab.table_type = 'BASE TABLE'
      and tab.table_schema not in ('pg_catalog',
          'information_schema')
      and tco.constraint_name is null
order by table_schema,
         table_name;
```

The result lists all schemas and for each schema the tables that don't have a primary key. Ignore the **pglogical** schema. Write down all the databases and tables listed in the schemas that aren't **pglogical**. At this point, the **notes** table in the **dmspg_1** schema and in the **dmspg_2** doesn't have a primary key.

8. Exit the PostgreSQL shell:

```
exit;
```

At this point, you have prepared the source instance and the source databases. They are ready for migration by Database Migration Service.

Connecting to the instance from the IDE

All instructions in this section are optional. Only follow them if you would like to connect to the instance and its databases using an IDE like DBeaver (https://dbeaver.io/). The instructions open the firewall for the IP address of your client device and configure PostgreSQL to accept connection requests from it.

1. Determine the IP address of your device. Search (https://www.google.com/search?q=what+is+my+ip+address) for pages that display your IP address and write it down (unless you know the device's IP address already).

2. In Cloud Shell, run the following command to open the firewall for your IP address and TCP port `5432`:

```
gcloud compute firewall-rules create my-device \
    --allow tcp:5432 \
    --source-ranges=device-ip
```

   Replace the following: ***device-ip***: your device's IP address

3. Use SSH to connect to the VM that runs the PostgreSQL instance.
4. Add your device IP address to the list of devices from which connections are allowed:
   a. Open the `pg_hba.conf` file, which is typically in the `/etc/postgresql/13/main` directory.
   b. Add a line that contains your device IP address to the `# IPv4 local connections` section.
   c. Copy the line that allows the IP address `127.0.0.1` to connect.
   d. Paste the line into the configuration file.
   e. Modify it to include your device's IP address.
   f. Terminate the IP address with `/32`.
5. Add the addresses that the PostgreSQL instance should listen to:
   a. Open the `postgresql.conf` file, which is typically in the `/etc/postgresql/13/main` directory.
   b. Find the commented entry `listen_addresses`.
   c. Modify it to specify the addresses that the PostgreSQL instance should listen to.
   d. Set the address to `'*'` so that any connection is accepted:

```
listen_addresses = '*'
```

   This change requires you to restart (https://www.postgresql.org/docs/current/runtime-config-connection.html) the source instance.

6. Restart the source instance:

   ```
   sudo systemctl restart postgresql@13-main
   ```

   After the instance restart, it's possible to connect to the databases in the instance from your device.

Later, Database Migration Service publishes the IP address of the target instance that needs to access the source instance to perform the data migration. To avoid restarting the source instance again, you can wait to perform the steps that open the source instance to your device until that point. Then, restarting the instance once accomplishes both changes at the same time (or even includes the configuration changes you made earlier).

If you opened the source instance to `'*'`, then another target instance restart isn't required. If you listed specific IP addresses, wait until the target instance IP address is available to avoid one instance restart.

All source instance and database preparations are now complete. You can now create a source instance connection profile. After that's complete, migrate the PostgreSQL instance.

Creating a source connection profile

Before creating a database migration job, create a source instance connection profile.

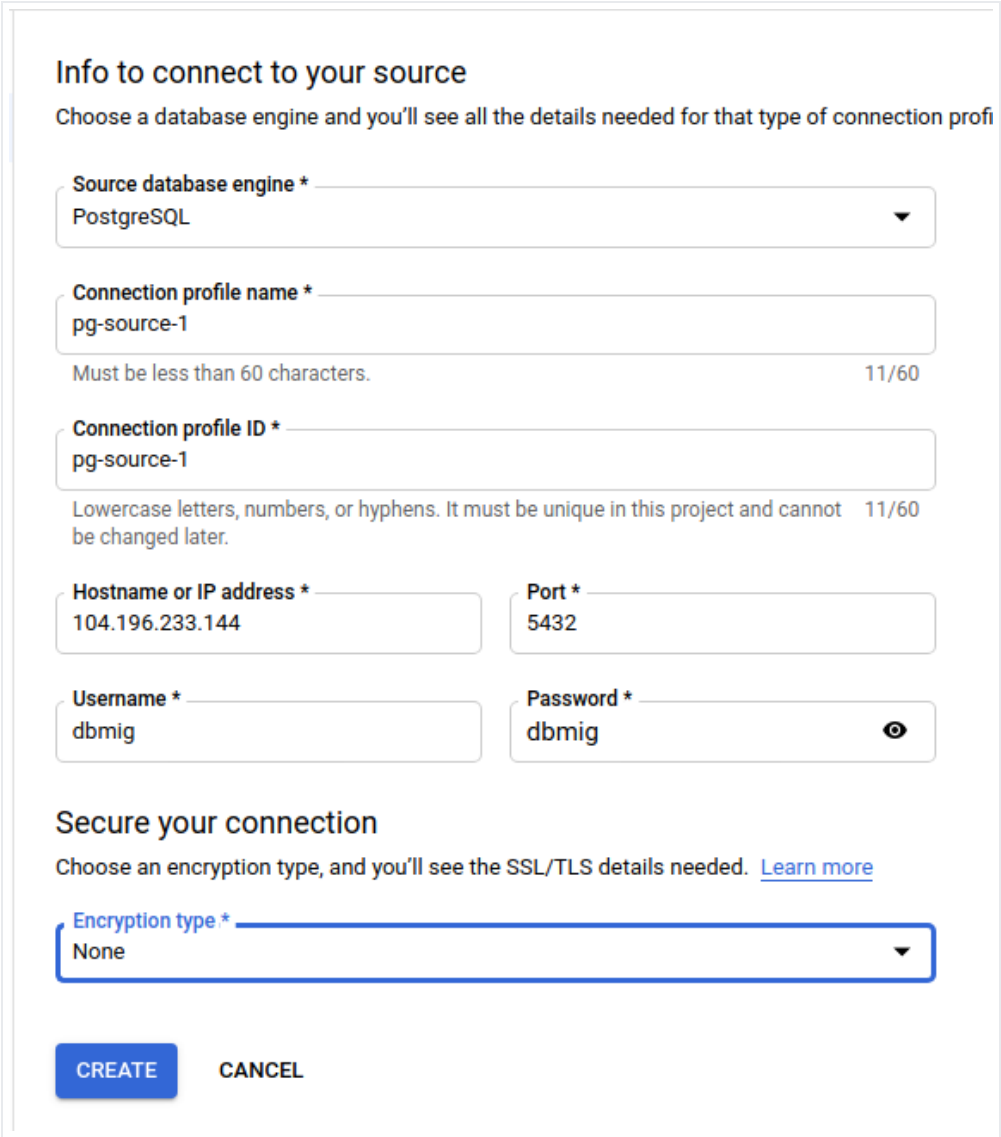
The following instructions are based on the Cloud Console user interface.

1. In the Cloud Console, go to the **Database Migration** page.

   Go to Database migration (https://console.cloud.google.com/dbmigration/migrations)

2. Select **Connection profiles** and then click **Create Profile**.
3. From the drop-down list, select **PostgreSQL**.
4. Fill in the fields:
   - **Connection profile name:** Use the same name as the compute instance.

- **Hostname or IP address:** Use the public IP address of the Compute Engine instance that runs your PostgreSQL instance.
- **Username:** Use the database migration user that you created earlier.



5. Click **Create**.
6. Check that the connection profile ID you created appears in the list of connection profiles.

Now that a connection profile exists that corresponds to the source PostgreSQL instance, you can refer to the profile by its name.

Specifying a database migration job

To specify a database migration job, complete the following steps:

1. In Cloud Console, go to the **Database Migration** page.

   Go to Database migration (https://console.cloud.google.com/dbmigration/migrations)

2. Select **Migration jobs** and then click **Create Migration Job**.
3. Review the steps you must perform to specify a migration job.

   If you want to read more, you can find additional information in Specifying a database migration job (/database-migration/docs/postgres/create-migration-job).

Describe your migration job

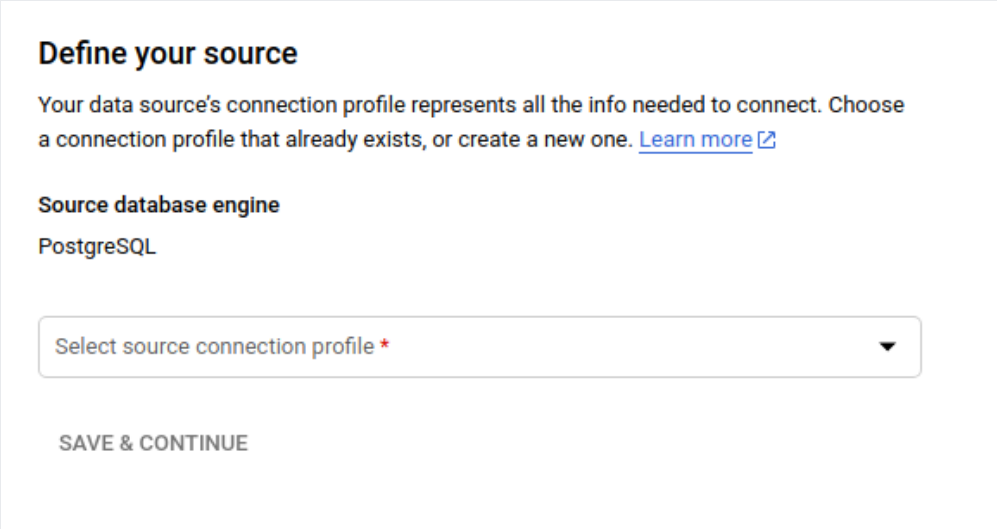
This step collects basic configuration information.

1. In Cloud Console, fill in the various input fields:
   a. **Migration job name:** Provide a name for the migration job. A best practice is to append some numbering scheme to the name or some other indicator. You might have to specify a few migration jobs for your testing purposes.
   b. **Source database engine:** Select PostgreSQL as the source database engine.
   c. **Destination region:** Select the destination region for the target Cloud SQL for PostgreSQL instance.
   d. **Migration job type:** Select the migration type that you want to perform from the drop-down list.
   e. Review the prerequisites to be sure that you meet all requirements.
2. Click **Save & Continue**.

Define a source

This step configures the source instance connection. The screen appears when you complete the previous step.

1. In Cloud Console, use the following screen to define a source:



The drop-down list shows the available connection profiles. It also gives you the option to create a connection profile.

2. Select the `pg-source-1` connection profile that you created earlier. The screen expands.
3. Click **Save & Continue**.

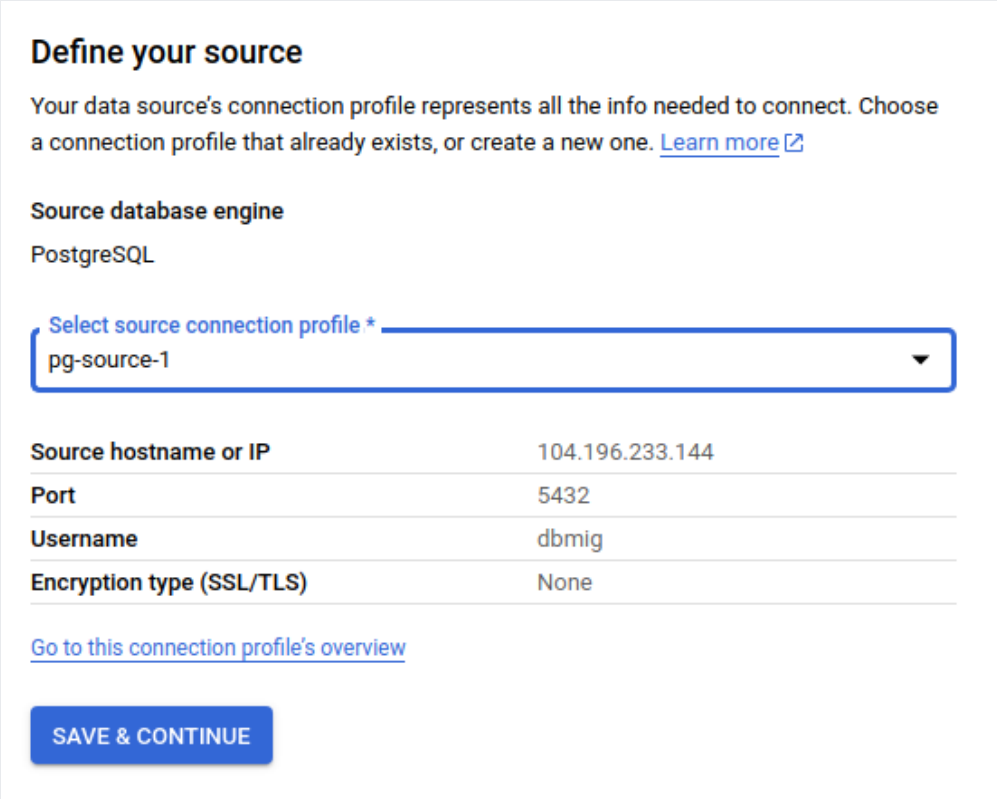
Create a destination

In this section, you configure the target instance.

1. In Cloud Console, navigate to the Database Migration page.

   Go to Database migration (https://console.cloud.google.com/dbmigration/migrations)

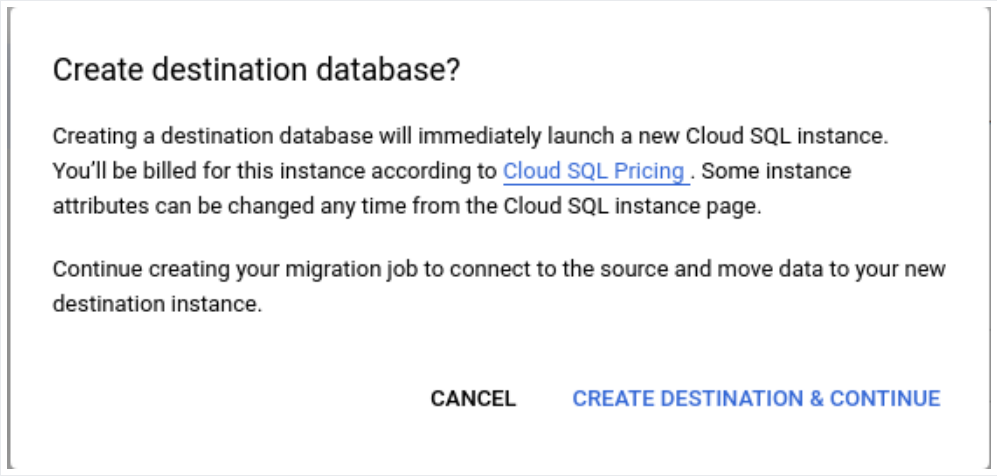
2. Create a destination. Provide a name, a root password, and select a target instance version. Ensure that you record the root password for future use.



★ **Important:** Write down your root password. It's required to log in to the target instance.

You can't change the instance name after you create it. The name you provide is the name of the new primary instance after the migration completes. The best practice is to name it for its role as a primary instance, not for its temporary role as a migration target (as we have done in this lesson). That name was chosen so you could better understand. In a production environment, however, the name you choose should reflect it being a (future) primary instance.

3. Select the appropriate IP address option for your destination instance. The options are **Private** or **Public** (/vpc/docs/ip-addresses).
4. Configure the target instance with the same settings you used for the source instance. In a production environment you would use the configuration values as determined by the test environment, as discussed in the Preparing a database migration (#preparing_a_database_migration) section.
5. Continue the configuration by selecting a storage type and storage capacity.
6. Add optional **Flag** and **Label** configurations if needed.
7. Click **Create & Continue**. The following dialog appears:



8. Click **Create Destination & Continue**. Creating the destination takes a few minutes.

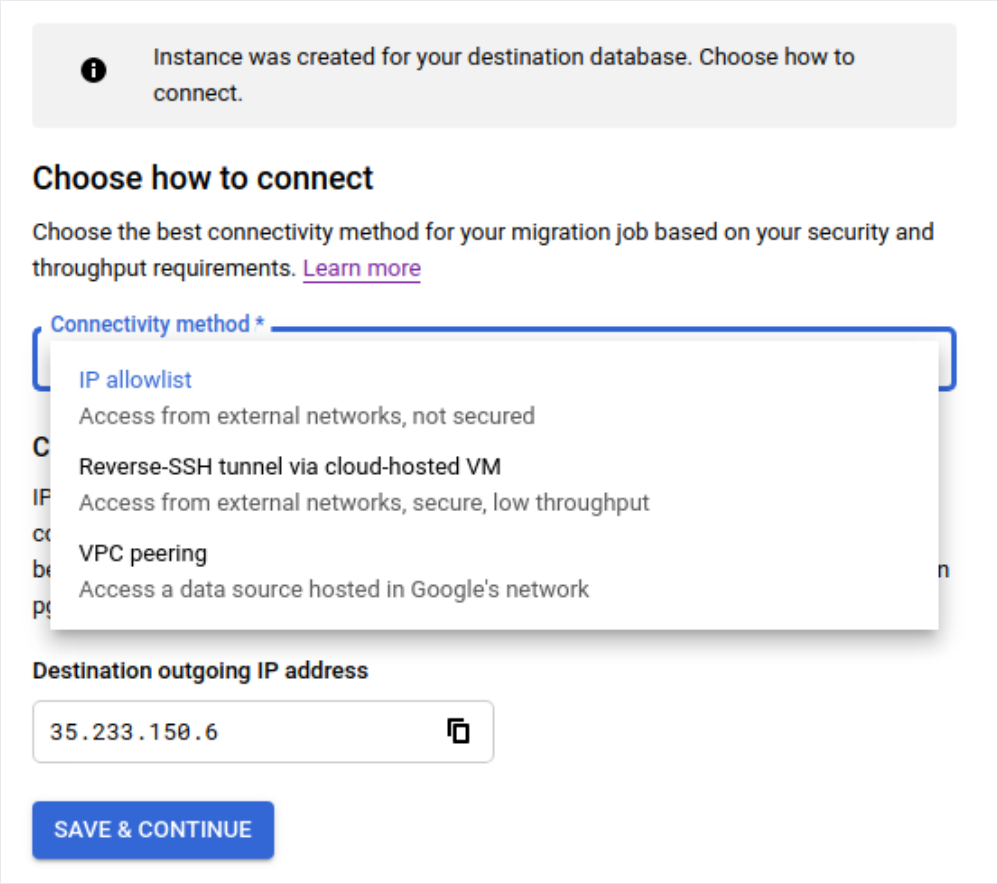
Define a connectivity method

In this section, you configure the connection to the source instance. After the target instance is created, its outgoing IP address appears. The source instance must be able to accept connections from this IP address.

1. In Cloud Console, navigate to the Database Migration page.

   Go to Database migration (https://console.cloud.google.com/dbmigration/migrations)

2. Select an option from the **Connectivity method** menu to update the source instance configuration. Base your selection on the config-file type you're using (`pg_hba.conf` and/or `postgresql.conf`):



Depending on how the source instance is configured you might have to restart it. This section is the place for the optimization that was discussed in Restart the source instance (#restart_the_source_instance). This restart enables all the configuration changes you made up to now to become effective.

If you only need to modify `pg_hba.conf`, then reloading the configuration is sufficient. If you need to change `listen_addresses` in `postgresql.conf`, then you must restart the source instance.

3. Add a firewall rule to allow incoming traffic from the outgoing IP address of the target instance on port `5432` (or any other port that you might have configured):

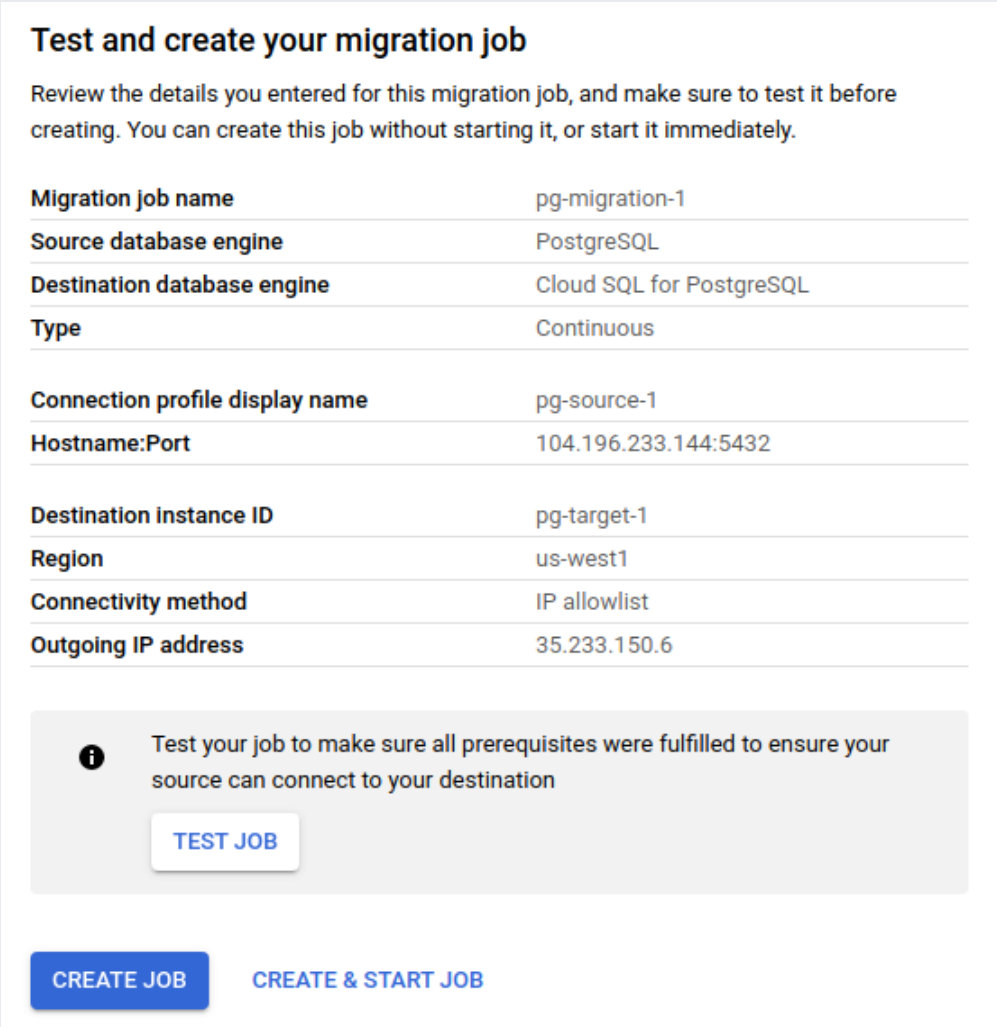
```
gcloud compute firewall-rules create pg-target-1 \
    --allow tcp:5432 \
    --source-ranges=<device-ip>
```

4. Click **Save & Continue**.

Test and create a migration job

This final step lets you test the configuration and save or start the migration job. While not required, configuration testing is a best practice.

1. In Cloud Console, click **Test Job**:



   Testing might take a while.

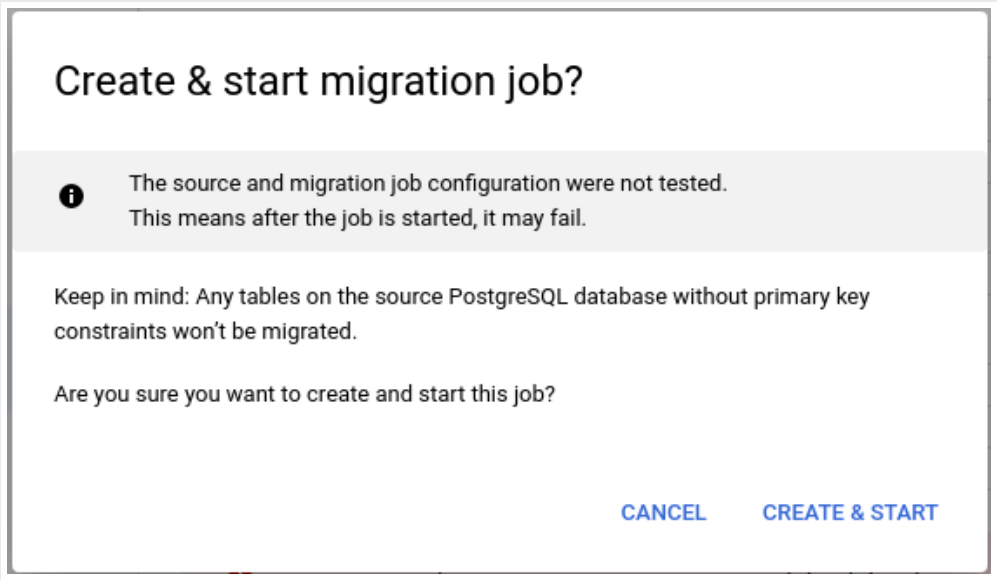
   If an error occurs, you see a warning message that defines the problem.

   For more information on debugging your errors see, Postgresql: Connection refused. (https://stackoverflow.com/questions/20825734/postgresql-connection-refused-check-that-the-hostname-and-port-are-correct-an)

2. When the test succeeds, a message appears saying *Tests passed successfully! You can create this job without starting it or start it immediately*.

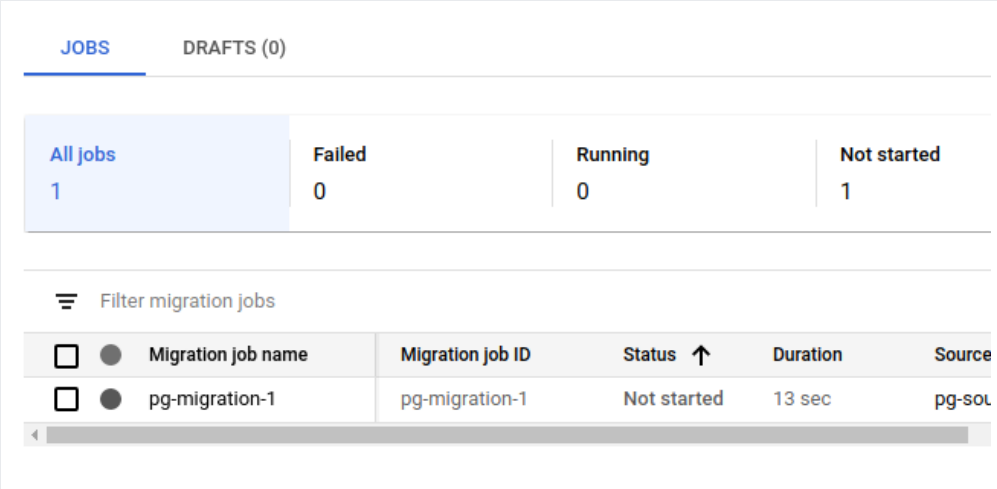
3. Click **Create Job**. If you have not tested, the system displays a warning:

4. Click **Cancel** to cancel out of the message.
5. Click **Create & Start Job**. A different warning message displays:



6. Click **Create & Start** to start the migration if you are certain that the source databases are ready to migrate.

   If you aren't certain, or if you have to make further preparations—for example non-primary key table preparation (#migrating-tables-without-pk)—cancel out of this error message, click **Save Job**, and then click **Create** in the **Create migration job** dialog. This action brings you back to the **Database Migration** page with the new migration job listed:



Preparing to run the database migration job

Before starting a database migration job, you must decide how to migrate tables without primary keys (non-PK tables). In addition, you must be aware of how to deal with materialized views, schema changes, and other aspects during migration.

The next sections discuss data types, database behavior, and how to make the target databases be 100% consistent with the source databases. Skip the paragraphs that don't apply to you.

Migrating tables without a primary key

When you start a migration job that contains one or more tables without a primary key, you see a warning that says *Keep in mind: Any tables on the source PostgreSQL database without primary key constraints won't be migrated. Are you sure you want to start the job?* After you've started, however, there are no additional warnings or reports on tables without primary keys (non-PK tables). Be aware that Database Migration Service doesn't move tables that don't have primary keys (/database-migration/docs/postgres/migration-fidelity#what_isnt_migrated). Manage these types of tables yourself.

You have to decide on the best time to manually migrate the tables without primary keys. Consider these two options:

- **Prepare non-PK tables before starting a migration job**. If you know that a non-PK table won't change until the migration completes, you can copy the data into a table with the same columns plus a surrogate primary key. This table is called a *transfer table*. This

approach migrates the data to the target database in the transfer table. Before the cutover, you can create the corresponding non-PK table in the target database and copy the data from the transfer table into the non-PK table. You can apply this approach to all non-PK tables where the rows won't change during the migration.

- **Migrate non-PK tables during the migration job execution**. If you know that a non-PK table changes during the migration, then you can create a transfer table with the same columns plus a surrogate primary key before you start the migration. This migration choice helps ensure that the migration job knows about this transfer table. However, you should only copy the data from the non-PK table into the transfer table when you know its schema and data won't change. The latest point to copy the data is when all data has been migrated, and shortly before the cutover.

These two approaches are outlined in the following sections. An alternative approach is to migrate non-PK tables without the help of Database Migration Service using the import facility of Cloud SQL (/sql/docs/postgres/import-export/importing).

Migrate non-PK tables before starting the database migration

To migrate non-PK tables of the source databases:

1. In Cloud Console, use SSH to connect to the Compute Engine instance.
2. Log in to the PostgreSQL shell:

```
sudo -u postgres psql
```

3. Create a transfer table. In this case, a transfer table is created for the non-PK table `notes`:

```
CREATE TABLE notes_transfer (
    note VARCHAR(256),
    surrogate_id SERIAL PRIMARY KEY);
```

4. Insert the rows of the non-PK table into its transfer table `notes_transfer`:

```
INSERT INTO notes_transfer (note)
    SELECT note FROM notes;
```

5. Select the rows to check that the insert worked:

   ```
   SELECT * FROM notes;
   ```

6. Ensure that the migration user has all necessary privileges for this new table and sequence as well (use **\dn** to find the relevant schema):

   ```
   GRANT SELECT on ALL TABLES in SCHEMA <schema> to dbmig;
   GRANT SELECT on ALL SEQUENCES in SCHEMA <schema> to dbmig;
   ```

7. Exit the PostgreSQL shell:

   ```
   exit;
   ```

   When the migration starts, the transfer table is migrated like any other regular table with a primary key.

After the migration completes, insert trasfer-table data into the various target databases and then drop the transfer table:

1. In Cloud Console, log in to the target replica following this procedure: Check the migration status in the target instance (#check-target-migration-status).

   The non-PK table was automatically created, but no data was migrated.

2. Insert the data from the transfer table into the non-PK table:

   ```
   INSERT INTO notes (note)
       SELECT note FROM notes_transfer;
   ```

3. Drop the transfer table:

   ```
   DROP TABLE notes_transfer;
   ```

Now the data in the non-PK tables of the source databases exists in the target databases.

Migrate non-PK tables during database migration

The steps in this section are the same as the previous section. However, inserting the data from the non-PK table into its corresponding transfer table occurs after the migration starts. Only start the migration when you are sure that the data in the non-PK table won't change.

If the data on the source changes in the non-PK table, you can delete all rows from the transfer table, and reinsert the data from the non-PK table. This action gives you the ability to correct the contents without having to start the migration from the beginning.

Manage materialized views

Database Migration Service migrates the schema of materialized views in a database, but not the data. See the bullet point on materialized views in the product documentation (/database-migration/docs/postgres/migration-fidelity#what_isnt_migrated) for more information.

1. To list all materialized view names, run:

   ```
   SELECT schemaname, matviewname
   FROM pg_matviews;
   ```

2. Run the following command before the application cutover for each materialized view in every target database:

   ```
   REFRESH MATERIALIZED VIEW <view_name>
   ```

   This command ensures that the materialized views are refreshed on the target databases.

Manage schema changes

Schema changes on source databases aren't automatically migrated by Database Migration Serviceto the target databases. Migrate these changes manually, as outlined in What changes are replicated during continuous migration (/database-migration/docs/postgres/faq#replicated) and in Continuous migration: PostgreSQL (/database-migration/docs/postgres/migration-fidelity#postgresql).

To make a schema change in a source database, follow these steps:

1. Stop all DML (database manipulation language) in the source database and wait for all data from the source database to migrate to the target database. This ensures that both the source and target database are quiet.
2. Change the schema on the source database and the target database.
3. Complete the changes in both the source database and target database before resuming any DML on the source database.

You have two options to implement and execute DDL (data definition language) statements:

- Use commands provided by the `pglogical` command `pglogical.replicate_ddl_command`. See Continuous migration: PostgreSQL (/database-migration/docs/postgres/migration-fidelity#postgresql) for examples.
- Execute the DML statements directly without the `pglogical` command. This method is preferred if you use a schema management tool. The only caveat is that the tool needs to be able to implement the same change twice: once on the source database and once on the target database.

Grant the role before running DDL on the target database (/database-migration/docs/postgres/migration-fidelity#postgresql).

If you have tables without primary keys, and you use the transfer table approach, any change to the base table must be applied to the transfer table.

Indirectly related to the execution of DDL statements is the following caveat. During the initial load, the DDL statements might be blocked if they can't access a required lock. Diagnose issues for PostgreSQL (/database-migration/docs/postgres/diagnose-issues) has a more detailed description of an error that might occur on a source database.

Manage large objects

`pglogical` can't migrate large objects (https://www.postgresql.org/docs/current/largeobjects.html). See What isn't migrated: PostgreSQL (/database-migration/docs/postgres/migration-fidelity#what_isnt_migrated) for more information. While `pglogical` doesn't migrate the rows of tables that contain large objects, a table is created in the target database. If you have tables with large objects, you must transfer them yourself, outside of Database Migration Service.

One approach to transfer large objects is to use `pg_dump` to export the table or tables that contain the large objects and import them into Cloud SQL. This export must be done separately for each database in the instance.

The following describes the process at a high level. The best time to execute these steps is after the target instance is available for import and after promotion. For each source and target database, determine all tables that contain large objects.

1. To try out the process (if you want), in Cloud Console, create an example table named `image` with one row:

   ```
   CREATE TABLE image (
       name text,
       raster oid);

   INSERT INTO image (name, raster)
       VALUES ('beautiful image', lo_import('<path>/image.jpg'));
   ```

   The insert statement adds one row to the table that contains an image file. See SQL-Oriented Large Object Functions (https://www.postgresql.org/docs/current/lo-funcs.html) to better understand the basic SQL commands for large objects.

2. Use `pg_dump` to extract the table or several tables:

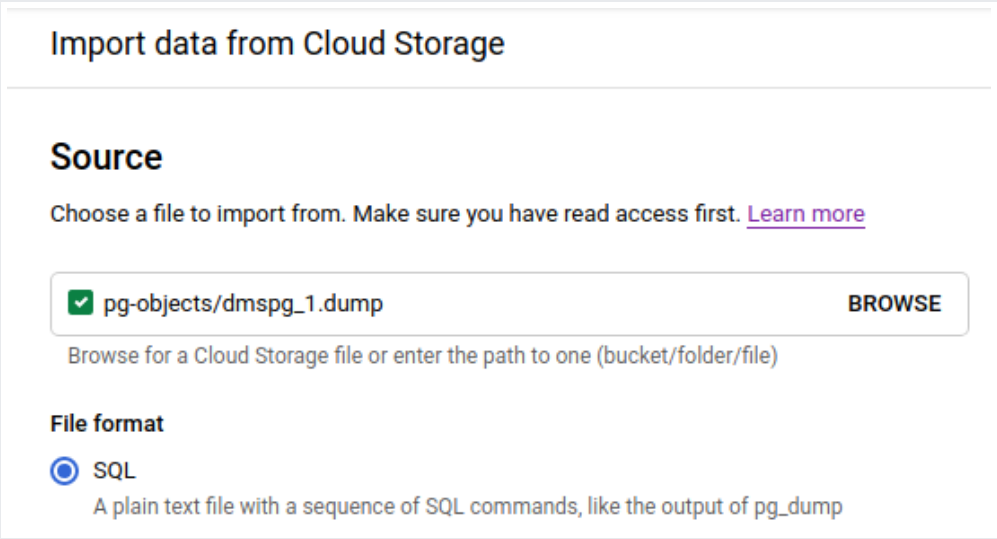
   ```
   sudo pg_dump --blobs -t image -h localhost --username=postgres dmspg_1 > <pa
   ```

   This command dumps the table `image` to a file called `dmspg_1.dump`.

3. Transfer the `dmspg_1.dump` file from the source system to a Cloud Storage bucket:

```
sudo gsutil cp dmspg_1.dump gs://pg-objects
```

4. Import the dump:



5. Check if the table exists and is populated after it imports:

```
select * from image;
```

6. Display some initial bytes and compare them to the target, to ensure that the image imported:

```
select lo_get(16951, 0, 1000);
```

   While this process is manual and has to be executed for each source and target database, it lets you migrate tables with large objects.

   You can perform the dump and import steps at any time. The safest time to perform these steps is when you know that the tables containing large objects won't change.

Manage foreign data and extensions

PostgreSQL has a set of extensions (https://www.postgresql.org/docs/current/external-extensions.html). PostgreSQL provides these

extensions; there is also an ecosystem of extensions available from other organizations—foreign data wrappers (https://www.postgresql.org/docs/current/ddl-foreign-data.html), for example.

For each extension that you use in the source instance, first ensure that the extension or an equivalent is available in Cloud SQL for PostgreSQL (/sql/docs/postgres/extensions). If Cloud SQL for PostgreSQL doesn't provide the extension, you must review the source instance and its application clients to see if it's possible to remove the extension from the source instance.

Any data managed by extensions outside of PostgreSQL isn't migrated. You must migrate those datasets independent of Database Migration Service.

Manage sequences

While Database Migration Service migrates sequences, the value of a sequence in the target might be different from the value of the sequence in the source (/database-migration/docs/postgres/migration-fidelity#what_isnt_migrated). In general, sequences in the target have a larger value than those in the source.

- To list all sequences in a database:

```
SELECT schemaname, sequencename
FROM pg_sequences;
```

- To check for the last value of a sequence (this example uses the sequence `public.notes_transfer_surrogate_id_seq`):

```
SELECT "last_value", log_cnt, is_called
FROM public.notes_transfer_surrogate_id_seq;
```

  To select the value of the sequence on the source, use this command. To select the value on the target, change the parameters and run the command again. If it's important that a sequence has the same value in the source and target, run the `ALTER SEQUENCE` (https://www.postgresql.org/docs/current/sql-altersequence.html) statement that best fits your situation.

Manage database users

Database Migration Service doesn't automatically move the database users of the source instance and databases, as outlined in the Database Migration Service documentation (/database-migration/docs/postgres/migration-fidelity#what_isnt_migrated).

- To list all users in a PostgreSQL instance, run the following command at the PostgreSQL command prompt:

```
\du
```

You must create the users yourself on the target instance. See Creating and managing PostgreSQL users (/sql/docs/postgres/create-manage-users) for more instructions.

Starting the database migration job

After you have decided how to migrate tables without primary keys, you can start the migration job. Follow these steps:

1. In Cloud Console, go to the **Database Migration** page.

   Go to Database migration (https://console.cloud.google.com/dbmigration/migrations)

2. Select the migration job from the list of migration jobs by selecting the checkbox associated with it.
3. Click the **Start** button, or select **Start** from the drop-down list on the right.
   - If an error occurs, a dialog explains what happened.
   - As the migration begins, the **Starting** dialog appears.
   - After the migration starts, the **Running • Full dump in progress** dialog appears.
   - While the migration job is running, the **Running • CDC in progress** dialog appears.

At this point, the migration job is running and data is migrating from the source databases to the target databases.

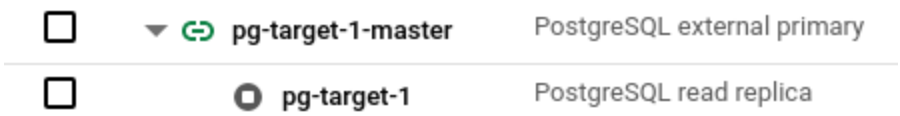
Check the migration status in the target instance

During the migration, you can log in to the target instance and check on the migration progress by connecting to the various databases and selecting data from tables.

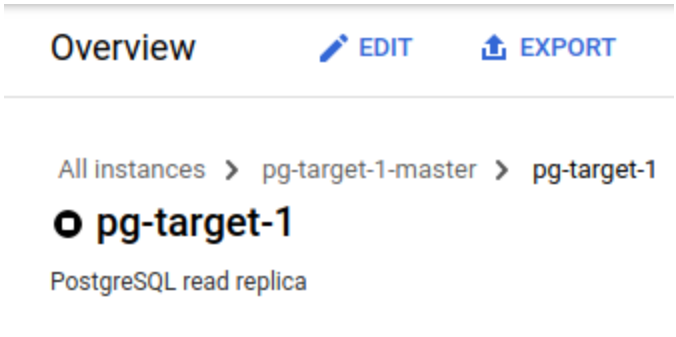
1. In Cloud Console, go to the Cloud SQL page:

   Go to Cloud SQL (https://console.cloud.google.com/sql/instances)

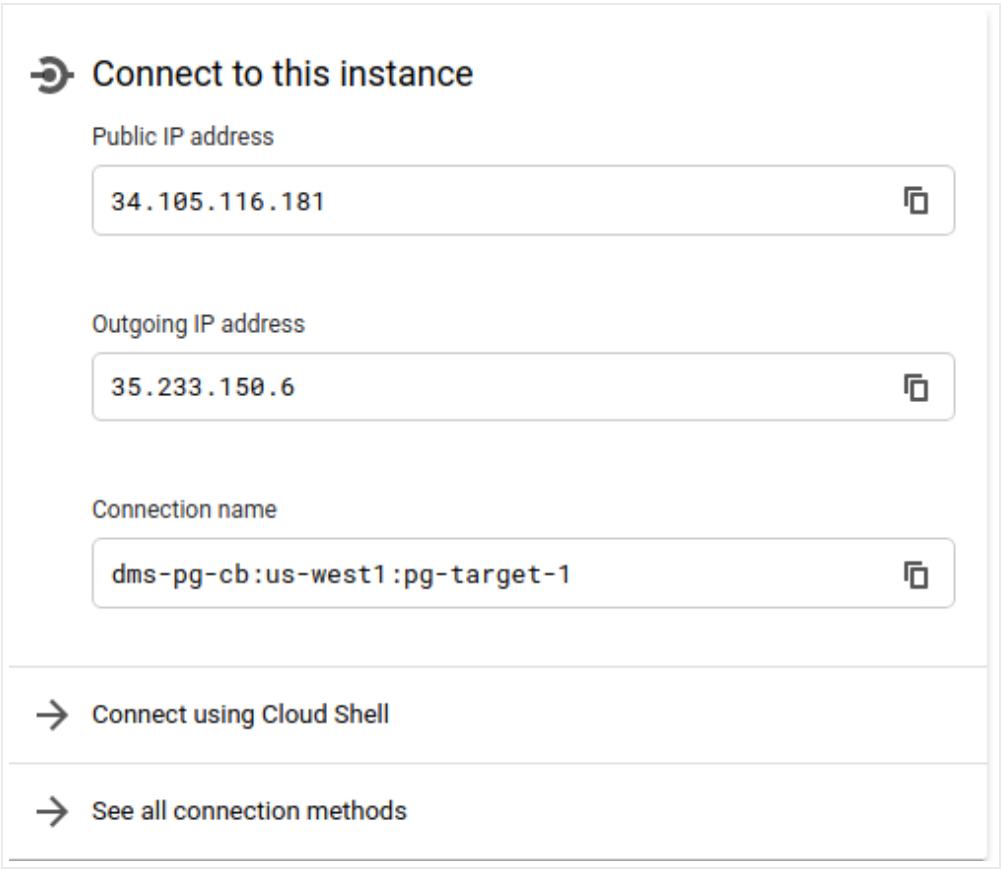
2. Select the replica that represents the target instance:



3. Select the name (in this case `pg-target-1`) to view the instance page:



4. Scroll to the **Connect to this instance** section:



5. Select **Connect using Cloud Shell**. This action opens Cloud Shell. From there, you can issue the usual commands to connect to the databases in the target instance. Use the root password that you provided when specifying the target instance details for the migration job.

   It might be necessary for you to enable the Cloud SQL Admin API at your first sign in. If you get a sign-in error, enable the Cloud SQL Admin API (/sql/docs/postgres/admin-api).

After you are connected, you can list the databases, connect to those and select data from the various tables.

Quiescing, data validation, promotion, application cutover, and database tuning

To continue to migrate data changes from the source databases to the target databases, the migration job has to be running.

At some point, unless you want to keep the migration running indefinitely, you have to use the target database as the new primary database. The following sections discuss the major steps to accomplish that goal.

Incurring downtime for quiescing and data validation

Database Migration Service provides a minimum-downtime migration by migrating data while applications use the source databases. However, to promote the target instance to be the new primary instance, you must end changes to the source databases so that Database Migration Service can migrate over any remaining changes.

In order to stop changes on the source databases, shut down all clients. This is called *quiescing* the source database. After changes to the source databases are complete, Database Migration Service migrates the remaining changes, ending the migration.

The most reliable way to determine that all data has been migrated is to make one last change manually on one of the source databases, and wait for that change to appear in the corresponding target database.

After the migration is complete you can validate that all data has been migrated correctly. One possible approach to this optional step is to randomly select a few tables, and run a count query on the source table and on the corresponding target table. The counts must be equal.

If you want to be thorough, write a script that compares the count of all tables in all databases. If you want to go further, you could execute aggregation queries for tables with columns that can be aggregated. In the extreme case, you would check that each row in the source and target rows are equivalent.

To help ensure that the database migration didn't accidentally insert rows that violate table constraints, you could validate that all table constraints are satisfied.

The more validation you do the longer the source and target instances will be unavailable. The least amount of downtime is when you don't validate, and you trust that the replication based on `pglogical` works correctly. The longest downtime is when you establish equivalence on a row by row basis across all tables of all databases.

Promote the migration job

After you quiesce the source database and after you perform all the validation you were planning, it's time to promote the migration job. Promoting a migration job stops the migration and promotes the target instance from being a replica to being a primary instance.

Promoting the migration job has no effect on the source instance. If for any reason any database's content is changed in the source instance, that change won't be migrated. The migration job is complete.

To promote a migration job, follow these instructions:

1. In Cloud Console, go to the **Database Migration** page.

   Go to Database migration (https://console.cloud.google.com/dbmigration/migrations)

2. Select the name of the migration job (once the pointer hovers over the name, it changes to a link). The **Details** page appears:
3. Select the **Promote** button. A dialog appears:
4. Select **Promote.** The status of the migration job changes to **Running • Promote in progress**:
5. After the promotion finishes, the migration job is complete:

   The migration job promotion changes the target instance from a replica to a regular primary instance. `pg-target-1` was originally a replica of `pg-target-1-master` (representing the source instance), now it's a standalone primary instance.

   Since `pg-target-1-master` has served its purpose, you can delete it.

6. Select `pg-target-1-master` (it changes to a link once the pointer hovers over the name). This brings you to the details page.
7. Click **Delete** and follow the instructions in the dialog that appears.
8. After the instance is deleted, the instance is removed from the instances list.

Application cutover to new primary databases

From the moment the migration job promotion completes, the target instance is the primary instance. You now need to connect all applications to the new primary instance.

In principle, you don't have to migrate applications when you migrate the instance. You can migrate applications before the instance is migrated, while the instance is migrated, after the instance is migrated, or not migrate at all. There are different approaches that depend on the complexity of the applications, the availability of personnel, and the risk factors of a concurrent migration.

Independent of the decision of when to migrate the application (or applications if there are several), the application has to access the new primary instance instead of the source instance. Ideally the connection is accomplished by a configuration change, not by a change of the application code.

Changes for all clients in the source instance aren't migrated to the new primary instance. It's a best practice to establish an inventory of the clients of the source instance well ahead of the cutover. Doing so lets you determine that the client can actually be configured to use the new primary instance once it's available.

Instance and database tuning

While not strictly part of the database migration process, once you cutover the applications to the target Cloud SQL instance you might need to tune the instance.

See Configuring database flags (/sql/docs/postgres/flags) and Performance Tips (https://www.postgresql.org/docs/current/performance-tips.html) for more information.

Source instance management

After the migration job promotion completes, changes to any of the source databases in the source instance aren't migrated.

Consider managing the source instance by disabling any non-read access. This action helps ensure that no erroneous access takes place without being immediately detected.

For example, if applications are cut over by updating their configuration files with new database connectivity specification, it's entirely possible that one or more applications might not be updated by omission. The same can happen with scripts that are still connecting to the source instance.

The best time for disabling any non-read access (or all access) to the source instance is after quiescing completes. This action helps ensure that no changes take place and that the cutover

is consistent.

High availability and disaster recovery setup

The target instance configuration options in Database Migration Service don't let you set up an HA Cloud SQL for PostgreSQL instance or create read-replicas in the same or different regions.

If you need a HA instance, follow the instructions (/sql/docs/postgres/configure-ha#ha-existing) for after the target instance was promoted. If you require read replicas, follow the instructions (/sql/docs/postgres/replication) to set up read replicas. Setting up read replicas is only possible after the target instance has been promoted.

Cleaning up

To avoid incurring charges to your Google Cloud account for the resources used in this tutorial, either delete the project that contains the resources, or keep the project and delete the individual resources.

Caution: Deleting a project has the following effects:

- **Everything in the project is deleted.** If you used an existing project for this tutorial, when you delete it, you also delete any other work you've done in the project.
- **Custom project IDs are lost.** When you created this project, you might have created a custom project ID that you want to use in the future. To preserve the URLs that use the project ID, such as an `appspot.com` URL, delete selected resources inside the project instead of deleting the whole project.

If you plan to explore multiple tutorials and quickstarts, reusing projects can help you avoid exceeding project quota limits.

1. In the Cloud Console, go to the **Manage resources** page.

   Go to Manage resources (https://console.cloud.google.com/iam-admin/projects)

2. In the project list, select the project that you want to delete, and then click **Delete**.

3. In the dialog, type the project ID, and then click **Shut down** to delete the project.

What's next

- Learn more about Database Migration Service for PostgreSQL (/database-migration/docs/postgres).
- Learn more about the `gcloud` commands supported by Database Migration Service (/sdk/gcloud/reference/database-migration).
- Explore reference architectures, diagrams, tutorials, and best practices about Google Cloud. Take a look at our Cloud Architecture Center (/architecture).

Last updated 2021-07-29 UTC.