

Create and Manage Cloud SQL for PostgreSQL Instances

Task 1. Migrate a stand-alone PostgreSQL database to a Cloud SQL for PostgreSQL instance

In this task you must migrate the stand-alone PostgreSQL `orders` database running on the `postgres-vm` virtual machine to a Cloud SQL for PostgreSQL instance using a Database Migration Services continuous migration job and VPC Peering connectivity.

Prepare the stand-alone PostgreSQL database for migration

In this sub-task you must prepare the stand-alone PostgreSQL database so that it satisfies the requirements for migration by Database Migration Services.

To complete this sub-task you must complete the following steps:

1. Enable the Google Cloud APIs required for Database Migration Services. Database Migration Services require the **Database Migration API** and the **Service Networking API** to be enabled in order to function. You must enable these APIs for your project.
2. Upgrade the target databases on the `postgres-vm` virtual machine with the `pglogical` database extension.

3. You must install and configure the **pglogical** database extension on the stand-alone PostgreSQL database on the `postgres-vm` Compute Instance VM. The pglogical database extension package that you must install is named `postgresql-13-pglogical`.
4. To complete the configuration of the **pglogical** database extension you must edit the PostgreSQL configuration file `/etc/postgresql/13/main/postgresql.conf` to enable the **pglogical** database extension and you must edit the `/etc/postgresql/13/main/pg_hba.conf` to allow access from all hosts.
5. Create a dedicated user for database migration on the stand-alone database.
6. The new user that you create on the stand-alone PostgreSQL installation on the `postgres-vm` virtual machine must be configured using the following user name and password:

- **install the pglogical database extension and jquery**

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

- **Download and apply some additions to the PostgreSQL configuration files (to enable pglogical extension)**

```
sudo su - postgres -c "gsutil cp gs://cloud-training/gsp918/pg_hba_append.conf ."
sudo su - postgres -c "gsutil cp gs://cloud-training/gsp918/postgresql_append.conf ."
sudo su - postgres -c "cat pg_hba_append.conf >> /etc/postgresql/13/main/pg_hba.conf"
sudo su - postgres -c "cat postgresql_append.conf >> /etc/postgresql/13/main/postgresql.conf"
sudo systemctl restart postgresql@13-main
```

- **Apply required privileges to postgres and orders databases**

```
sudo su - postgres
psql
\c postgres;
CREATE EXTENSION pglogical;
\c orders;
CREATE EXTENSION pglogical;
```

- **Migration user name** : `Postgres Migration User`
- **Migration user password** : `DMS_1s_cool!`

7. Grant that user the required privileges and permissions for databases to be migrated. Database Migration Services require that the migration user has privileges to specific schemata and relations of the target databases for migration, in this case that is the `orders` database.

The Database Migration Service requires all tables to be migrated to have a primary key.

8. You must make sure that all of the tables in the `orders` database have a primary key set before you start the migration.

- `distribution_centers`
- `inventory_items`
- `order_items`
- `products`
- `users`

```
CREATE USER migration_user PASSWORD 'DMS_1s_cool!';
```

```
ALTER DATABASE orders OWNER TO migration_user;
```

```
ALTER ROLE migration_user WITH REPLICATION;
```

```
\c orders;
```

```
SELECT column_name FROM information_schema.columns WHERE table_name =  
'inventory_items' AND column_name = 'id';
```

```
ALTER TABLE inventory_items ADD PRIMARY KEY (id);
```

```
GRANT USAGE ON SCHEMA pglogical TO migration_user;
```

```
GRANT ALL ON SCHEMA pglogical TO migration_user;
```

```
GRANT SELECT ON pglogical.tables TO migration_user;
```

```
GRANT SELECT ON pglogical.depend TO migration_user;
```

```
GRANT SELECT ON pglogical.local_node TO migration_user;
```

```
GRANT SELECT ON pglogical.local_sync_status TO migration_user;
```

```
GRANT SELECT ON pglogical.node TO migration_user;  
GRANT SELECT ON pglogical.node_interface TO migration_user;  
GRANT SELECT ON pglogical.queue TO migration_user;  
GRANT SELECT ON pglogical.replication_set TO migration_user;  
GRANT SELECT ON pglogical.replication_set_seq TO migration_user;  
GRANT SELECT ON pglogical.replication_set_table TO migration_user;  
GRANT SELECT ON pglogical.sequence_state TO migration_user;  
GRANT SELECT ON pglogical.subscription TO migration_user;
```

```
GRANT USAGE ON SCHEMA public TO migration_user;  
GRANT ALL ON SCHEMA public TO migration_user;  
GRANT SELECT ON public.distribution_centers TO migration_user;  
GRANT SELECT ON public.inventory_items TO migration_user;  
GRANT SELECT ON public.order_items TO migration_user;  
GRANT SELECT ON public.products TO migration_user;  
GRANT SELECT ON public.users TO migration_user;
```

```
ALTER TABLE public.distribution_centers OWNER TO migration_user;  
ALTER TABLE public.inventory_items OWNER TO migration_user;  
ALTER TABLE public.order_items OWNER TO migration_user;  
ALTER TABLE public.products OWNER TO migration_user;  
ALTER TABLE public.users OWNER TO migration_user;
```

```
\c postgres;
```

```
GRANT USAGE ON SCHEMA pglogical TO migration_user;

GRANT ALL ON SCHEMA pglogical TO migration_user;

GRANT SELECT ON pglogical.tables TO migration_user;

GRANT SELECT ON pglogical.depend TO migration_user;

GRANT SELECT ON pglogical.local_node TO migration_user;

GRANT SELECT ON pglogical.local_sync_status TO migration_user;

GRANT SELECT ON pglogical.node TO migration_user;

GRANT SELECT ON pglogical.node_interface TO migration_user;

GRANT SELECT ON pglogical.queue TO migration_user;

GRANT SELECT ON pglogical.replication_set TO migration_user;

GRANT SELECT ON pglogical.replication_set_seq TO migration_user;

GRANT SELECT ON pglogical.replication_set_table TO migration_user;

GRANT SELECT ON pglogical.sequence_state TO migration_user;


GRANT SELECT ON pglogical.subscription TO migration_user;
```

Task 2. Promote a Cloud SQL to be a stand-alone instance for reading and writing data

- In this task, you must complete the migration by promoting the Cloud SQL for PostgreSQL instance to a stand-alone instance.
- When the promotion is complete, the status of the job updates to Completed.

Create a new continuous migration job

In this step you will create a new continuous migration job.

1. In the Google Cloud Console, on the **Navigation menu** () , click **VIEW ALL PRODUCTS** under Databases section click on **Database Migration > Migration jobs**.
2. Click + **Create Migration Job**.
3. For **Migration job name**, enter **vm-to-cloudsql**.
4. For **Source database engine**, select **PostgreSQL**.
5. For **Destination region**, select **(region)**.
6. For **Destination database engine**, select **Cloud SQL for PostgreSQL**.
7. For **Migration job type**, select **Continuous**.

Leave the defaults for the other settings.

8. Click **Save & Continue**.

Define the source instance

In this step, you will define the source instance for the migration.

1. For **Source connection profile**, select **postgres-vm**.
Leave the defaults for the other settings.

2. Click **Save & Continue**.

Create the destination instance

In this step, you will create the destination instance for the migration.

1. For **Destination Instance ID**, enter **postgresql-cloudsql**.
2. For **Password**, enter **supersecret!**.
3. For **Choose a Cloud SQL edition**, select **Enterprise** edition.
4. For **Database version**, select **Cloud SQL for PostgreSQL 13**.
5. In **Choose region and zone** section, select **Single zone** and select **(zone)** as **primary zone**.
6. For **Instance connectivity**, select **Private IP** and **Public IP**.
7. Select **Use an automatically allocated IP range**.

Leave the defaults for the other settings.

8. Click **Allocate & Connect**.

Associated networking VPC to peer

Select a network to create a private connection.

VPC *
default ▼

If you plan on connecting to the migration source via VPC peering, choose the VPC where it resides.

Managed services network creation ?

☐ Select IP range



You don't have any allocated ranges. Allocate a new custom IP range. [Learn more](#)

☒ Use an automatically allocated IP range

ALLOCATE & CONNECT

Note: This step may take a few minutes. If asked to retry the request, click the Retry button to refresh the Service Networking API.

When this step is complete, an updated message notifies you that the instance will use the existing managed service connection.

You will need to edit the `pg_hba.conf` file on the VM instance to allow access to the IP range that is automatically generated in point 5 of the previous step. You will do this in a later step before testing the migration configuration at the end of this task.

Associated networking VPC to peer

Select a network to create a private connection.

VPC *
default ▼

If you plan on connecting to the migration source via VPC peering, choose the VPC where it resides.



This instance will use the existing managed service connection.

Enter the additional information needed to create the destination instance on Cloud SQL.


9. For **Machine shapes**, check **1 vCPU, 3.75 GB**

10. For **Storage type**, select **SSD**
11. For **Storage capacity**, select **10 GB**
12. Click **Create & Continue**.

If prompted to confirm, click **Create Destination & Continue**. A message will state that your destination database instance is being created. Continue to the next step while you wait.

Define the connectivity method


In this step, you will define the connectivity method for the migration.

 Instance creation in progress. This may take a few minutes. You can begin defining the connectivity method. Configure & Continue is enabled when both are complete.

1. For **Connectivity method**, select **VPC peering**.
2. For **VPC**, select **default**.

VPC peering is configured by **Database Migration Service** using the information provided for the VPC network (the default network in this example).

When you see an updated message that the destination instance was created, proceed to the next step.

 Instance was created. Define the connectivity method, then Configure & Continue.

3. Click **Configure & Continue**.

Allow access to the postgresql-vm instance from automatically allocated IP range

In this step you will edit the `pg_hba.conf` PostgreSQL configuration file to allow the Database Migration Service to access the stand-alone PostgreSQL database.

1. Get the allocated IP address range. In the Google Cloud Console on the **Navigation menu** (≡), right-click **VPC network** > **VPC network peering** and open it in a new tab.
2. Click on the `servicenetworking-googleapis-com` entry and then click on **Effective Routes View** at the bottom.
3. From the dropdown for **Network** select **default** and for **Region** select `(region)`. Click **View**.
4. In the **Destination IP range** column, copy the IP range (e.g. 10.107.176.0/24) next to **peering-route-xxxxx...** route.
5. In the Terminal session on the VM instance, edit the `pg_hba.conf` file as follows:

```
sudo nano /etc/postgresql/13/main/pg_hba.conf
```

Copied!

content_copy

6. On the last line of the file:

```
#GSP918 - allow access to all hosts
host    all all 0.0.0.0/0    md5
```

replace the "all IP addresses" range (0.0.0.0/0) with the range copied in point 3 above.

```
#GSP918 - allow access to all hosts
host    all all 10.107.176.0/24    md5
```

Note: The above step is not required to make the migration work, but it is good practice to make the source database more secure during the migration process, and also restricts access after the migration when the migrated database becomes the source of truth.

7. Save and exit the nano editor with Ctrl-O, Enter, Ctrl-X
8. Restart the PostgreSQL service to make the changes take effect. In the VM instance Terminal session:

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

Copied!

content_copy

Test and start the continuous migration job

In this step, you will test and start the migration job.

1. In the **Database Migration Service** tab you open earlier, review the details of the migration job.
2. Click **Test Job**.
3. After a successful test, click **Create & Start Job**.

If prompted to confirm, click **Create & Start**.

Task 3. Implement Cloud SQL for PostgreSQL IAM database authentication

In this task you must configure the newly migrated Cloud SQL for PostgreSQL instance to support Cloud IAM users and IAM database authentication. You are also required to patch the Cloud SQL for PostgreSQL instance, to add the public ip-address of the postgres-vm virtual machine to the list of networks that are allowed to connect to the instance.

To complete this task you must complete the following steps:

1. Patch the Migrated Cloud SQL for PostgreSQL Instance ID Cloud SQL instance to allow connections from the public ip-address of the postgres-vm virtual machine.
 - In the Migrated Cloud SQL for PostgreSQL Instance ID Cloud SQL instance, go to **connections > Networking**.
 - Under the Public IP, click on **ADD A NETWORK**. For the network, use the external IP of the postgres-vm virtual machine.
2. In the Migrated Cloud SQL for PostgreSQL Instance ID Cloud SQL instance, create a Cloud SQL IAM user using the lab student ID, Qwiklabs user account name, as the principal account name.
 - Click **Users > Add user account**, then select **Cloud IAM**.
3. Grant SELECT permission to the Cloud IAM user for the orders table.
 - In the Migrated Cloud SQL for PostgreSQL Instance ID Cloud SQL instance, go to **Overview**. Under **Connect to this instance**, click on **Open Cloud Shell**.
 - For the password enter supersecret!. Then connect to the orders database using `\c orders;` command.
 - Again for the password enter supersecret!.
 - Use the following command to grant SELECT permission. Replace the **Table_Name** and **Qwiklabs_User_Account_Name** variables with the correct values.

```
GRANT SELECT ON Table_Name TO "Qwiklabs_User_Account_Name";
```


Copied!

content_copy
4. Run the following query as the Qwiklabs user account name user in the migrated database to confirm that the Qwiklabs user account name can select data from the orders table.
5.

```
SELECT COUNT(*) FROM orders
```

Connect to the PostgreSQL instance

1. In the **Replica Instance** menu, click **Overview**.
2. Scroll down to the **Connect to this instance** section and click **Open Cloud Shell**.

The command to connect to PostgreSQL will pre-populate in Cloud Shell:

```
gcloud sql connect postgresql-cloudsql --user=postgres --quiet
```

3. Run the pre-populated command.
If prompted, click **Authorize** for the API.

4. When prompted for a password, which you previously set, enter:
supersecret!

You have now activated the PostgreSQL interactive console for the destination instance.

Review the data in the Cloud SQL for PostgreSQL instance

1. To select the database in the PostgreSQL interactive console, run the following command:
`\c orders;`
Copied!

content_copy

2. When prompted for a password, enter:
supersecret!

- ⚠ **Change the TABLE_NAME and USER_NAME by given lab instructions**

```
GRANT ALL PRIVILEGES ON TABLE [TABLE_NAME] TO "USER_NAME";
```

```
\q
```

Task 4. Configure and test point-in-time recovery

In this task you must configure point-in-time recovery on a Cloud SQL for PostgreSQL instance and then test it by using point-in-time recovery to create a cloned instance at a point in time that rolls back some changes.

To complete this task you must complete the following steps:

1. Enable backups on the Cloud SQL for PostgreSQL instance.
 - In the Migrated Cloud SQL for PostgreSQL Instance ID Cloud SQL instance, go to **Overview**. Click on **edit > Data Protection**.
 - Enable point-in-time recovery and set the number of retained transaction log days to Point-in-time recovery retention days.
2. Make a note of the timestamp for the point-in-time you wish to restore to using the following command.

```
date -u --rfc-3339=ns | sed -r 's/ /T/; s/\.([0-9]{3})\.*/\.\1Z/'
```

Copied!

content_copy

3. Make some changes to the database after this timestamp.
 - In the Migrated Cloud SQL for PostgreSQL Instance ID Cloud SQL instance, go to **Overview**. Under **Connect to this instance**, click on **Open Cloud Shell**.
 - For the password enter supersecret!. Then connect to the orders database using `\c orders;` command.
 - Again for the password enter supersecret!.
 - You must add a row of data to the `orders.distribution_centers` table.
4. Use point-in-time recovery to create a clone that replicates the instance state at your chosen timestamp.
 - Use the following command to clone the instance by replacing the **CLOUDSQL_INSTANCE**, **NEW_INSTANCE_NAME** and **TIME_STAMP** variables with the correct values.
 - ```
gcloud sql instances clone $CLOUDSQL_INSTANCE
$NEW_INSTANCE_NAME \
--point-in-time $TIME_STAMP
```

Copied!

content\_copy

- For the **Cloned instance name** you must use the name `postgres-orders-pitr`.

The new instance will not be used, but do not discard it, as it will be required to confirm that you have correctly completed the lab.

**Note:** You must specify the point-in-time recovery timestamp in UTC time, specified in RFC 3339 format 'yyyy-MM-ddThh:mm:ss.mmmZ'.

```
date --rfc-3339=seconds
```

Copy the given output and Save this

- **Asking For a password enter**

```
supersecret!
```

Copy and paste the password and the password will not visible to you

```
\c orders
```

- **Asking For a password enter**

```
supersecret!
```

Copy and paste the password and the password will not visible to you

```
insert into distribution_centers values(-80.1918,25.7617,'Miami FL',11);
\q
```

```
gcloud auth login --quiet
```

```
gcloud projects get-iam-policy $DEVSHHELL_PROJECT_ID
```

```
export INSTANCE_ID=
```

```
gcloud sql instances clone $INSTANCE_ID postgres-orders-pitr --point-in-time
'CHANGE_TIMESTAMP'
```