# Securing a Cloud SQL for PostgreSQL Instance

## Task 1. Create a Cloud SQL for PostgreSQL instance with CMEK enabled

In this task you will create a Cloud SQL for PostgreSQL instance with CMEK enabled. It is imperative that you keep the keys safe as you cannot manage your database without them.

### Create a per-product, per-project service account for Cloud SQL

You can create the service account you require for Cloud SQL CMEK using the `gcloud beta services identity create` command.

1. In Cloud Shell, run:

```
export PROJECT_ID=$(gcloud config list --format 'value(core.project)')
gcloud beta services identity create \
    --service=sqladmin.googleapis.com \
    --project=$PROJECT_ID
```

2. Click the **Authorize** button if prompted.

This creates the service account that you will bind to the CMEK in a later step.

### Create a Cloud Key Management Service keyring and key

In this step you will create a Cloud KMS keyring and key to use with CMEK.

1. In Cloud Shell, to create the Cloud KMS keyring, run:

```
export KMS_KEYRING_ID=cloud-sql-keyring
export ZONE=$(gcloud compute instances list --filter="NAME=bastion-vm"
--format=json | jq -r .[].zone | awk -F "/zones/" '{print $NF}')
export REGION=${ZONE::-2}
gcloud kms keyrings create $KMS_KEYRING_ID \
    --location=$REGION
```

2. In Cloud Shell, to create the Cloud KMS key, run:

```
export KMS_KEY_ID=cloud-sql-key
gcloud kms keys create $KMS_KEY_ID \
 --location=$REGION \
 --keyring=$KMS_KEYRING_ID \
 --purpose=encryption
```

3. In Cloud Shell, to bind the key to the service account, run:

```
export PROJECT_NUMBER=$(gcloud projects describe ${PROJECT_ID} \
    --format 'value(projectNumber)')
gcloud kms keys add-iam-policy-binding $KMS_KEY_ID \
    --location=$REGION \
    --keyring=$KMS_KEYRING_ID \
    --member=serviceAccount:service-${PROJECT_NUMBER}@gcp-sa-cloud-
sql.iam.gserviceaccount.com \
    --role=roles/cloudkms.cryptoKeyEncrypterDecrypter
```

The service account name is the same name that was returned by the `gcloud beta services identity create` command in the previous sub-task.

## Create a Cloud SQL instance with CMEK enabled

In this step you will create a Cloud SQL for PostgreSQL instance with CMEK enabled. It is not possible to patch an existing instance to enable CMEK, so you should bear this in mind if you plan to use CMEK to encrypt your data.

In order to access your Cloud SQL instance from external development or application environments, you can configure the Cloud SQL instance with a public IP address and control access by allowlisting the IP address of those environments. This limits access to the public interface to those address ranges that you specify.

You will treat the Compute Engine VM instance in the lab as a development environment and will therefore need the to allow list the external IP address of that instance. You will also add the external IP address of the Cloud Shell to the allowlist to make it easier to complete tasks later in the lab.

1. In Cloud Shell, find the external IP address of the `bastion-vm` VM instance:

```
export AUTHORIZED_IP=$(gcloud compute instances describe bastion-vm \
    --zone=$ZONE \
    --format 'value(networkInterfaces[0].accessConfigs.natIP)')
echo Authorized IP: $AUTHORIZED_IP
```

2. In Cloud Shell, find the external IP address of the Cloud Shell:

```
export CLOUD_SHELL_IP=$(curl ifconfig.me)
echo Cloud Shell IP: $CLOUD_SHELL_IP
```

3. In Cloud Shell, create your Cloud SQL for PostgreSQL instance with:

```
export KEY_NAME=$(gcloud kms keys describe $KMS_KEY_ID \
    --keyring=$KMS_KEYRING_ID --location=$REGION \
    --format 'value(name)')

export CLOUDSQL_INSTANCE=postgres-orders
gcloud sql instances create $CLOUDSQL_INSTANCE \
    --project=$PROJECT_ID \
    --authorized-networks=${AUTHORIZED_IP}/32,$CLOUD_SHELL_IP/32 \
    --disk-encryption-key=$KEY_NAME \
    --database-version=POSTGRES_13 \
    --cpu=1 \
    --memory=3840MB \
    --region=$REGION \
    --root-password=supersecret!
```

4. Enter 'y' if asked after entering the command.

# Task 2. Enable and configure pgAudit on a Cloud SQL for PostgreSQL database

In this task you will enable and configure the pgAudit database extension which enables fine-grained control of logging of all types of database activity.

1. In Cloud Shell enter the following to add the pgAudit database flags to your Cloud SQL instance:

```
gcloud sql instances patch $CLOUDSQL_INSTANCE \
    --database-flags cloudsql.enable_pgaudit=on,pgaudit.log=all
```

2. Hit enter to confirm and continue. Wait a minute for the patch command to complete before continuing.

3. In Cloud Console, on the **Navigation menu** (☰), click **Databases** > **SQL** and click on the Cloud SQL instance named `postgres-orders`.

4. In the Cloud SQL **Overview** panel, top menu, click **Restart** and **Restart** again in the pop-up dialog.

It will take a minute to restart your Cloud SQL for PostgreSQL instance, then continue below.

5. In Cloud Console, in the **Connect to this instance** section, click **Open Cloud Shell**. A command will be auto-populated to the Cloud Shell.

6. Run that command and enter the password `supersecret!` when prompted. A **psql** session will start in Cloud Shell.

7. In **psql**, to create the `orders` database and enable the pgAudit extension to log all reads and writes, run:

```
CREATE DATABASE orders;
\c orders;
```

8. Enter the password `supersecret!` again.

9. In **psql**, to create and configure the database extension, run:

```
CREATE EXTENSION pgaudit;
ALTER DATABASE orders SET pgaudit.log = 'read,write';
```

## Enable Audit Logging in Cloud Console

In this step you will enable Audit Logging in Cloud Console.

1. In Cloud Console, on the **Navigation menu** (☰), click **IAM & Admin** > **Audit Logs**.

2. In the **Filter** box, type `Cloud SQL` and select the entry in the drop-down list.

3. Check the checkbox for `Cloud SQL` on the left and all the checkboxes in the **Info Panel** on the right.

4. Click the **Save** button in the **Info Panel**.

## Populate a database on Cloud SQL for PostgreSQL

In this step you will populate the `orders` database with data provided to you.

1. Click the + icon on the Cloud Shell title bar to open a new tab in the Cloud Shell, and in that tab, to download the data and database population scripts, run:

```
export SOURCE_BUCKET=gs://cloud-training/gsp920
gsutil -m cp ${SOURCE_BUCKET}/create_orders_db.sql .
gsutil -m cp ${SOURCE_BUCKET}/DDL/distribution_centers_data.csv .
gsutil -m cp ${SOURCE_BUCKET}/DDL/inventory_items_data.csv .
gsutil -m cp ${SOURCE_BUCKET}/DDL/order_items_data.csv .
gsutil -m cp ${SOURCE_BUCKET}/DDL/products_data.csv .
gsutil -m cp ${SOURCE_BUCKET}/DDL/users_data.csv .
```

2. In the same Cloud Shell session, create and populate the database as follows:

```
export CLOUDSQL_INSTANCE=postgres-orders
export POSTGRESQL_IP=$(gcloud sql instances describe $CLOUDSQL_INSTANCE
--format="value(ipAddresses[0].ipAddress)")
export PGPASSWORD=supersecret!
psql "sslmode=disable user=postgres hostaddr=${POSTGRESQL_IP}" \
    -c "\i create_orders_db.sql"
```

It will take a minute for the `orders` database to be populated.

3. Exit the terminal session in the new tab:

```
Exit
```

4. In your **psql** session in the remaining Cloud Shell tab, to further log all `SELECT` operations on a particular relation, for example the `order_items` table, run the following commands:

```
CREATE ROLE auditor WITH NOLOGIN;
ALTER DATABASE orders SET pgaudit.role = 'auditor';
GRANT SELECT ON order_items TO auditor;
```

5. Run the first `SELECT` query below :

Summary of orders by usersSummary by individual productOrders by distribution center

```
SELECT
    users.id  AS users_id,
    users.first_name  AS users_first_name,
    users.last_name  AS users_last_name,
    COUNT(DISTINCT order_items.order_id ) AS order_items_order_count,
    COALESCE(SUM(order_items.sale_price ), 0) AS
order_items_total_revenue
FROM order_items
LEFT JOIN users ON order_items.user_id = users.id
GROUP BY 1, 2, 3
ORDER BY 4 DESC
LIMIT 500;
```

6. The output is 500 rows long, so hit `q` and the colon prompt to return to the `orders=>` prompt.

7. Repeat the last two steps for the other two queries tabs in the code block.

8. Exit **psql**:

```
\q
```

## View pgAudit logs

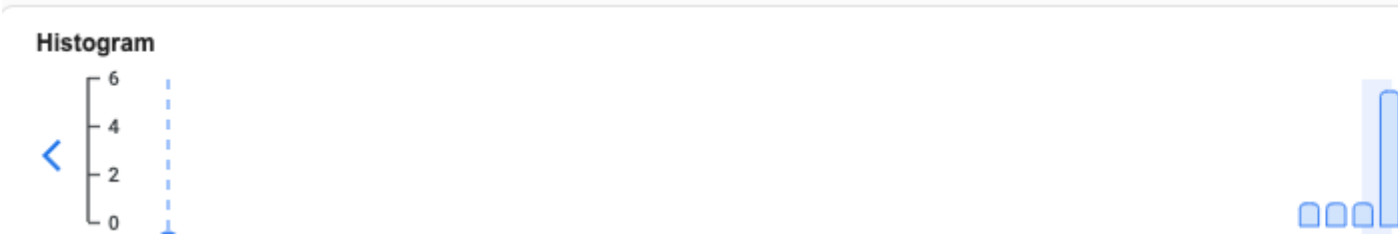
In this step you will view the logging of your database updates and queries in the pgAudit logs.

1. In Cloud Console, on the **Navigation menu** (☰), click **Observablity** > **Logging** > **Logs Explorer**.

2. In the **Query** tab of the **Logs Explorer**, paste the following and click the **Run query** button:

```
resource.type="cloudsql_database"
logName="projects/qwiklabs-gcp-01-
edcd3adb0ddc/logs/cloudaudit.googleapis.com%2Fdata_access"
protoPayload.request.@type="type.googleapis.com/google.cloud.sql.audit.
v1.PgAuditEntry"
```

3. In the histogram displayed, you can see the audit activity associated with your DDL inserts and the `SELECT` queries you ran earlier.




4. Click on the last bar on the histogram - this corresponds to the `SELECT` queries you ran.

In the **Query results** panel below the histogram, the log entries are listed.

5. Expand a log entry, and under `protoPayload.request` you will see the `SELECT` query.

# Task 3. Configure Cloud SQL IAM database authentication

In this task you will configure Cloud SQL IAM database authentication. All of the database access and update tasks you have performed so far have used built-in PostgreSQL user accounts. You can also create Cloud SQL for PostgreSQL users using Cloud IAM accounts. Database users can authenticate to Cloud SQL using Cloud IAM instead of using built-in database accounts and fine-grained permissions at the database level can be granted to those users.

In this task you will configure the lab user account as a Cloud SQL IAM user, grant that user access to the `orders.order_items` database table using the **postgres** administrator account, and then test access to the `orders.order_items` database table from the command line using the **psql** command line utility.

The authentication process that is used in this task is explained in detail in the IAM authentication documentation for Cloud SQL for PostgreSQL.

## Test database access using a Cloud IAM user before Cloud SQL IAM authentication is configured.

You will attempt to access the database using a Cloud IAM user before Cloud SQL IAM authentication haas been enabled in order to establish that the Cloud IAM user cannot initially access the data.

- Test access to the `orders` database using the lab student account as the username:

```
export USERNAME=$(gcloud config list --format="value(core.account)")
export CLOUDSQL_INSTANCE=postgres-orders
export POSTGRESQL_IP=$(gcloud sql instances describe $CLOUDSQL_INSTANCE
--format="value(ipAddresses[0].ipAddress)")
export PGPASSWORD=$(gcloud auth print-access-token)
psql --host=$POSTGRESQL_IP $USERNAME --dbname=orders
```

This connection attempt will fail and you will see an authentication failed message similar to the following because the Cloud SQL IAM user has not been created yet:

```
psql --host=$POSTGRESQL_IP $USERNAME --dbname=orders
psql: error: connection to server at "35.226.251.234", port 5432
failed: FATAL:  password authentication failed for user "student-01-
22fa974575e4@qwiklabs.net"
connection to server at "35.226.251.234", port 5432 failed: FATAL:
password authentication failed for user "student-01-
22fa974575e4@qwiklabs.net"
```

Cloud SQL IAM database authentication uses OAuth 2.0 access tokens is the Cloud IAM user password, which are short-lived and only valid for one hour so you should regenerate the token every time you need to authenticate. The access token should always be passed into the **psql** command using the **PGPASSWORD** environment variable as the buffer for the **psql** password parameter is too small to hold the OAuth 2.0 token string.

## Create a Cloud SQL IAM user

You will now create a Cloud SQL IAM user and confirm that Cloud SQL IAM user authentication has been enabled.

1. In Cloud Console, on the **Navigation menu** (☰), click **Databases** > **SQL** and click on the Cloud SQL instance named `postgres-orders`.

2. In the Cloud SQL **Overview** panel, in the **Configuration** panel note that the **Database flags** list includes **pgAudit.log** and **cloudsql.enable_pgaudit** only.

3. Click **Users** to open the **Users** panel.

4. Click **Add user account**.

5. Select **Cloud IAM**.

6. In the **Principal** box enter the lab student name.

7. Click **Add**.

8. Wait for the new user to be added.

9. Click **Overview** and now note that **cloudsql.iam_authentication** has been added to the list of database flags.

## Grant the Cloud IAM user access to a Cloud SQL database table

You will now connect to the `postgres-orders` instance using the built in `postgres` administrator account and grant access to the `orders.order_items` table to the Cloud IAM user.

1. In the Cloud Shell connect to the `postgres-orders` Cloud SQL instance as the `postgres` administrative user:

```
gcloud sql connect postgres-orders --user=postgres -quiet
```

2. When prompted enter the password: `supersecret!`.

3. Enter the following SQL command to switch to the `orders` database:

```
\c orders
```

When prompted for a password enter `supersecret!` again.

4. Enter the following SQL command to grant the lab user all permissions on the `order_items` table. The Cloud IAM username for the lab has been inserted into this query for you.

```
GRANT ALL PRIVILEGES ON TABLE order_items TO "student-00-
13748dd6f5b0@qwiklabs.net";
\q
```

## Test database access using a Cloud IAM user after Cloud SQL IAM authentication is configured.

You will repeat the attempt to access the database using a Cloud IAM user after Cloud SQL IAM authentication has been enabled in order to establish that the Cloud IAM user can now access the data.

You can now test access to the database again using the Cloud IAM user instead of the built-in `postgres` user:

1. In the Cloud Shell enter the following command to connect to the database using the Cloud IAM database user:

```
export PGPASSWORD=$(gcloud auth print-access-token)
psql --host=$POSTGRESQL_IP $USERNAME --dbname=orders
```

This connection succeeds and you are now connected the instance using Cloud IAM user authentication.

2. Test your access permission by running this query:

```
SELECT COUNT(*) FROM order_items;
```

This will now return a successful result:

```
orders=> SELECT COUNT(*) FROM order_items;
 count
--------
 198553
(1 row)
```

3. Confirm that you do not have access to one of the other tables by running this query:

```
SELECT COUNT(*) FROM users;
```

This will now return a successful result:

```
orders=> SELECT COUNT(*) FROM users;
ERROR:  permission denied for table users
```