MySQL (/sql/docs/mysql/connect-functions) | **PostgreSQL** | SQL Server

ature is in a pre-release state and might change or have limited support. For more information, see the product launch stages lucts/#product-launch-stages).

This page contains information and examples for connecting to a Cloud SQL instance from a service running in Cloud Functions.

Cloud SQL is a fully-managed database service that makes it easy to set up, maintain, manage, and administer your relational PostgreSQL and MySQL databases in the cloud.

Cloud Functions (/functions/docs/concepts/overview) is a lightweight compute solution for developers to create single-purpose, stand-alone functions that respond to Cloud events without the need to manage a server or runtime environment.

1. Create a Cloud SQL for PostgreSQL instance (/sql/docs/postgres/create-instance).

   ★ **Note:** These instructions require your Cloud SQL instance to have a public IP address. If you want to use a private IP address, see Configuring Serverless VPC Access (/vpc/docs/configure-serverless-vpc-access) and connect directly using the private IP.

2. Find the `INSTANCE_CONNECTION_NAME` for the instance on the **Instance details** page. It uses the format `PROJECT_ID:REGION:INSTANCE_ID`, and is used to identify the Cloud SQL instance you are connecting to.
3. Enable the Cloud SQL Admin API, if you haven't already done so:

   Enable the API (https://console.cloud.google.com/flows/enableapi?apiid=sqladmin&redirect=https://console.cloud.google.com)

Cloud Functions does not require any special configuration beyond making sure the service account used has the correct permissions.

Cloud Functions uses a service account to authorize your connections to Cloud SQL. This service account must have the correct IAM permissions to successfully connect. Unless otherwise configured, the default service account is in the format `service-PROJECT_NUMBER@gcf-admin-robot.iam.gserviceaccount.com`.

When connecting resources in two different projects, make sure that both projects have enabled the correct IAM roles and have given the service account the correct permissions.

Ensure that the service account for your service has one of the following IAM roles (/iam/docs/understanding-roles#cloud-sql-roles):

- `Cloud SQL Client` (preferred)
- `Cloud SQL Editor`
- `Cloud SQL Admin`

Or, you can manually assign the following IAM permissions (https://cloud.google.com/storage/docs/access-control/using-iam-permissions):

- `cloudsql.instances.connect`
- `cloudsql.instances.get`

For detailed instructions on adding IAM roles to a service account, see Granting Roles to Service Accounts (/iam/docs/granting-roles-to-service-accounts).

Once correctly configured, you can connect your service to your Cloud SQL instance's unix domain socket using the format: `/cloudsql/INSTANCE_CONNECTION_NAME`.

These connections are automatically encrypted without any additional configuration.

The PostgreSQL standard requires a `.s.PGSQL.5432` suffix in the socket path. Some libraries apply this suffix automatically, but others
e you to specify the socket path as follows: `/cloudsql/INSTANCE_CONNECTION_NAME/.s.PGSQL.5432`.

ng: Linux based operating systems have a maximum socket path length of 107 characters. If the total length of the path exceeds this len
ll not be able to connect with a socket from Cloud Functions.

By default, Cloud Functions **does not** support connecting to the Cloud SQL instance using TCP. Your code should **not** try to access the ins
an IP address (such as `127.0.0.1` or `172.17.0.1`) unless you have configured Serverless VPC Access
'docs/configure-serverless-vpc-access).

You can use the Cloud SQL proxy (/sql/docs/postgres/sql-proxy) when testing your application locally. See the quickstart for using the proxy for local testing (/sql/docs/mysql/quickstart-proxy-test) for detailed instructions.

Connections to underlying databases may be dropped, either by the database server itself, or by the infrastructure underlying Cloud Functions. To mitigate this, we recommend that you use a client library that supports connection pools that automatically reconnect broken client connections.

Additionally, we recommend using a globally scoped connection pool as this improves the likelihood that your function reuses the same connection for subsequent invocations of the function, and closes the connection naturally when the instance is evicted (auto-scaled down).

For more detailed examples on how to use connection pools, see Managing database connections (/sql/docs/postgres/manage-connections).

Cloud SQL imposes a maximum limit on concurrent connections, and these limits may vary depending on the database engine chosen (see Cloud SQL Quotas and Limits (/sql/docs/quotas#fixed-limits)).

When using a connection pool, it is important to set the maximum connections to 1. This may seem counter-intuitive, however, Cloud Functions limits concurrent executions to 1 per instance. This means you never have a situation where two requests are being processed by a single function instance at the same time. This means in most situations only a single database connection is needed.

Where possible, you should take care to only initialize a connection pool for functions that need to use it. If a deployed function initializes a connection pool it doesn't need, it could create unused connections that count towards your quota. For more details on dealing with global variables in Cloud Functions, see Tips & Tricks (https://cloud.google.com/functions/docs/bestpractices/tips#use_global_variables_to_reuse_objects_in_future_invocations).

For more detailed examples on how to limit the number of connections, see Managing database connections (/sql/docs/mysql/manage-connections#count).