Using Cloud SQL for PostgreSQL

Python (https://cloud.google.com/appengine/docs/flexible/python/using-cloud-sql-postgres) | Java (https://cloud.google.com/appengine/docs/flexible/java/using-cloud-sql-postgres) | Node.js (https://cloud.google.com/appengine/docs/flexible/nodejs/using-cloud-sql-postgres) | Go (https://cloud.google.com/appengine/docs/flexible/go/using-cloud-sql-postgres) | Ruby (https://cloud.google.com/appengine/docs/flexible/ruby/using-cloud-sql-postgres) | PHP (https://cloud.google.com/appengine/docs/flexible/php/using-cloud-sql-postgres) | **.NET**

This page shows how to connect to a Cloud SQL for PostgreSQL instance from an App Engine application, and how to read and write to Cloud SQL. Cloud SQL is a SQL database that lives in Google's cloud.

To learn more about Cloud SQL, see the Cloud SQL documentation (https://cloud.google.com/sql/docs). For information on Cloud SQL pricing and limits, see the Cloud SQL Pricing page (https://cloud.google.com/sql/pricing). App Engine applications are also subject to the App Engine quotas (https://cloud.google.com/appengine/docs/quotas).

Before you begin

1. Create or select a Google Cloud project in the Cloud Console and then ensure that project includes an App Engine application and billing is enabled:

   GO TO APP ENGINE (HTTPS://CONSOLE.CLOUD.GOOGLE.COM/PROJECTSELECTOR/APPENGINE/CF

   The **Dashboard** opens if an App Engine application already exists in your project and billing is enabled. Otherwise, follow the prompts for choosing a region (https://cloud.google.com/appengine/docs/locations) and enabling billing.

2. Enable the Cloud SQL Admin API.

   ENABLE THE API (HTTPS://CONSOLE.CLOUD.GOOGLE.COM/FLOWS/ENABLEAPI?APIID=SQLADMIN

3. To deploy your app with the `gcloud` tool, you must download, install, and initialize the Cloud SDK:

DOWNLOAD THE SDK (HTTPS://CLOUD.GOOGLE.COM/SDK/DOCS/)

4. Install the .NET Core SDK, LTS version (https://www.microsoft.com/net/download/core#/lts).
5. If you are using Visual Studio, to build and run .NET core applications you must install .NET Core tools (https://www.microsoft.com/net/core#windowsvs2015).
6. If you are using Visual Studio, to make it easy to deploy to App Engine install Google Cloud Tools for Visual Studio (https://cloud.google.com/tools/visual-studio/docs/quickstart#install_cloud_tools_for_visual_studio).

Configuring the Cloud SQL instance

To create and configure a Cloud SQL instance:

1. Create a Cloud SQL for PostgreSQL instance (https://cloud.google.com/sql/docs/postgres/create-instance).
2. If you haven't already, set the password for the default user on your Cloud SQL instance:

```
gcloud sql users set-password postgres no-host --instance [INSTANCE_NAME] --pas
```

3. If you don't want to use the default user to connect, create a user (https://cloud.google.com/sql/docs/postgres/create-manage-users#creating).

Configure SSL access to the Cloud SQL instance

1. Follow instructions to create a client certificate and require SSL (https://cloud.google.com/sql/docs/mysql/configure-ssl-instance#new-client).

★ **Note:** Ensure that you configure the instance to require SSL connections. Otherwise, the security of your instance could be compromised.

2. From the Instance details page, click **Access Controls > Authorization**.
3. Click **+ Add Network**.
4. Enter `all` for the name.

5. Enter `0.0.0.0/0` for the network.
6. Click **Done**, then **Save**.
7. To generate a `client.pfx` file from the certificate files you created in step 1, enter at the command line:

   ```
   openssl pkcs12 -export -in client-cert.pem -inkey client-key.pem -certfile serv
   ```

   If you don't have a machine with openssl installed, use Cloud SDK (https://cloud.google.com/shell/docs/).
8. Replace the `client.pfx` file in the `dotnet-docs-samples\appengine\flexible\CloudSql` project with the `client.pfx` you created.

Setting the connection string and adding a library

Set up the local environment to support connections for local testing.

For example, for the provided code sample, add the connection string to the `appsettings.json` file.

The connection string includes the user, password, and IP address:

appengine/flexible/CloudSql/appsettings.json (https://github.com/GoogleCloudPlatform/dotnet-docs-samples/blob/master/appengine/flexible/CloudSql/appsettings.json)

ORM/DOTNET-DOCS-SAMPLES/BLOB/MASTER/APPENGINE/FLEXIBLE/CLOUDSQL/APPSETTINGS.JSON)

```
"ConnectionString": "Uid=aspnetuser;Pwd=;Host=cloudsql;Database=visitors"
```

The connection string is used to create the connection:

appengine/flexible/CloudSql/Startup.cs (https://github.com/GoogleCloudPlatform/dotnet-docs-samples/blob/master/appengine/flexible/CloudSql/Startup.cs)

JDPLATFORM/DOTNET-DOCS-SAMPLES/BLOB/MASTER/APPENGINE/FLEXIBLE/CLOUDSQL/STARTUP.CS)

```
var connectionString = new NpgsqlConnectionStringBuilder(
    Configuration["CloudSql:ConnectionString"])
```

```
{
    // Connecting to a local proxy that does not support ssl.
    SslMode = SslMode.Disable
};
NpgsqlConnection connection =
    new NpgsqlConnection(connectionString.ConnectionString);
```

Running the sample code

The following sample writes visit information to Cloud SQL and then reads and returns the last ten visits:

appengine/flexible/CloudSql/Controllers/HomeController.cs (https://github.com/GoogleCloudPlatform/dotnet-docs-samples/blob/master/appengine/flexible/CloudSql/Controllers/HomeController.cs)

S-SAMPLES/BLOB/MASTER/APPENGINE/FLEXIBLE/CLOUDSQL/CONTROLLERS/HOMECONTROLLER.CS)

```
// Insert a visit into the database:
using (var insertVisitCommand = _connection.CreateCommand())
{
    insertVisitCommand.CommandText =
        @"INSERT INTO visits (user_ip) values (@user_ip)";
    var userIp = insertVisitCommand.CreateParameter();
    userIp.ParameterName = "@user_ip";
    userIp.DbType = DbType.String;
    userIp.Value =
        FormatAddress(HttpContext.Connection.RemoteIpAddress);
    insertVisitCommand.Parameters.Add(userIp);
    await insertVisitCommand.ExecuteNonQueryAsync();
}

// Look up the last 10 visits.
using (var lookupCommand = _connection.CreateCommand())
{
    lookupCommand.CommandText = @"
        SELECT * FROM visits
        ORDER BY time_stamp DESC LIMIT 10";
    List<string> lines = new List<string>();
    var reader = await lookupCommand.ExecuteReaderAsync();
    HomeModel model = new HomeModel() {
        VisitorLog = new List<VisitorLogEntry>()
    };
    while (await reader.ReadAsync()) {
        model.VisitorLog.Add(new VisitorLogEntry() {
            IpAddress = reader.GetString(1),
            TimeStamp = reader.GetDateTime(0)
        });
    }
    return View(model);
}
```

Testing and deploying

VISUAL STUDIO COMMAND LINE

To test your application locally:

1. In Visual Studio, open `dotnet-docs-samples\appengine\flexible\AppEngineFlex.sln`.
2. Press `F5`.

To deploy your application:

1. In Solution Explorer, right-click **CloudSql**, and choose **Publish CloudSql to Google Cloud...**
2. Click **App Engine Flex**.
3. Click **Publish**.