Ruby (https://cloud.google.com/ruby/) Guides

Using Cloud SQL for PostgreSQL with Rails 5

Get started developing Ruby on Rails apps that run on the App Engine flexible environment (https://cloud.google.com/appengine/docs/flexible/). The apps you create run on the same infrastructure that powers all of Google's products, so you can be confident that they can scale to serve all of your users, whether there are a few or millions of them.

This tutorial assumes you are familiar with Rails web development. It walks you through setting up Cloud SQL for PostgreSQL (https://cloud.google.com/sql/docs/postgres/) with a new Rails app. You can also use this tutorial as a reference for configuring existing Rails apps to use Cloud SQL for PostgreSQL.

This tutorial requires Ruby 2.3.4 (https://www.ruby-lang.org/) or newer.

Before you begin

1. Sign in (https://accounts.google.com/Login) to your Google Account.

   If you don't already have one, sign up for a new account (https://accounts.google.com/SignUp).

2. In the 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 THE PROJECT SELECTOR PAGE (HTTPS://CONSOLE.CLOUD.GOOGLE.COM/PROJECTSELECT

3. Make sure that billing is enabled for your Google Cloud project. Learn how to confirm billing is enabled for your project (https://cloud.google.com/billing/docs/how-to/modify-project).
4. Install and initialize the Cloud SDK (https://cloud.google.com/sdk/docs/).
5. Enable the Datastore, Pub/Sub, Cloud Storage JSON, Stackdriver Logging, and Google+ APIs.

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

Preparing a Cloud SQL for PostgreSQL instance

Set up a Cloud SQL for PostgreSQL instance for this tutorial.

1. Create a PostgreSQL instance (https://cloud.google.com/sql/docs/postgres/create-instance). In this tutorial, the name of the instance is `rails-cloudsql-instance`.
2. Create a database in the instance (https://cloud.google.com/sql/docs/postgres/create-manage-databases). In this tutorial, the name of the production database is `cat_list_production`.
3. Set the postgres user password for the instance (https://cloud.google.com/sql/docs/postgres/create-manage-users#user-root).

Setting up your local environment for Rails

To set up your local environment for this tutorial:

1. Install Ruby (https://www.ruby-lang.org/) version 2.3.4 or newer.
2. Install the Rails 5 (http://guides.rubyonrails.org/) gem.
3. Install the Bundler (https://bundler.io/) gem.

Note: Alternatively, you can use Cloud Shell (https://console.cloud.google.com/?cloudshell=true), which comes with Ruby, Rails, Bundler, and the Cloud SDK already installed.

For additional information on installing Rails and its dependencies, see the official Getting started with Rails (http://guides.rubyonrails.org/getting_started.html) guide.

After you complete the prerequisites, create and deploy a Rails app by using Cloud SQL for PostgreSQL. The following sections guide you through configuring, connecting to Cloud SQL for PostgreSQL, and deploying an app.

Create a new app to list cats

1. Run the `rails new` command to create a new Rails app. This app stores a list of cats in Cloud SQL for PostgreSQL.

```
rails new cat_sample_app
```

2. Go to the directory that contains the generated Rails app.

```
cd cat_sample_app
```

Run the app locally

To run the new Rails app on your local computer:

1. Install dependencies by using Bundler.

```
bundle install
```

2. Start a local web server:

```
bundle exec bin/rails server
```

3. In a browser, go to http://localhost:3000/ (http://localhost:3000/)

A **Yay! You're on Rails!** message from the app displays on the page.

Note: To exit the web server, in your terminal window, press `Ctrl+C`.

Generate scaffolding for a list of cats

Generate scaffolding for a resource named `Cat` that is used to form a list of cats with their name and age.

1. Generate the scaffolding.

```
bundle exec rails generate scaffold Cat name:string age:integer
```

The command generates a model, controller, and views for the `Cat` resource.

```
invoke  active_record
create    db/migrate/20170804210744_create_cats.rb
create    app/models/cat.rb
invoke    rspec
create      spec/models/cat_spec.rb
```

```
invoke  resource_route
route    resources :cats
invoke  scaffold_controller
create    app/controllers/cats_controller.rb
invoke    erb
create      app/views/cats
create      app/views/cats/index.html.erb
create      app/views/cats/edit.html.erb
create      app/views/cats/show.html.erb
create      app/views/cats/new.html.erb
create      app/views/cats/_form.html.erb
invoke    jbuilder
create      app/views/cats/index.json.jbuilder
create      app/views/cats/show.json.jbuilder
create      app/views/cats/_cat.json.jbuilder
invoke  assets
invoke    js
create      app/assets/javascripts/cats.js
invoke    scss
create      app/assets/stylesheets/cats.scss
invoke  scss
create    app/assets/stylesheets/scaffolds.scss
```

2. Open the file `config/routes.rb` to see the following generated content.

appengine/rails-cloudsql-postgres/config/routes.rb (https://github.com/GoogleCloudPlatform/ruby-docs-samples/blob/0a6e898a1acd140ca04c63c7b68da1468acb4082/appengine/rails-cloudsql-postgres/config/routes.rb)

CD140CA04C63C7B68DA1468ACB4082/APPENGINE/RAILS-CLOUDSQL-POSTGRES/CONFIG/ROUTES.RB)

```
Rails.application.routes.draw do
  resources :cats
  get 'cats/index'
  # For details on the DSL available within this file, see http://guides.rubyon

end
```

3. Add `root 'cats#index'` to the file.

appengine/rails-cloudsql-postgres/config/routes.rb (https://github.com/GoogleCloudPlatform/ruby-docs-samples/blob/0a6e898a1acd140ca04c63c7b68da1468acb4082/appengine/rails-cloudsql-postgres/config/routes.rb)

CD140CA04C63C7B68DA1468ACB4082/APPENGINE/RAILS-CLOUDSQL-POSTGRES/CONFIG/ROUTES.RB)

```
Rails.application.routes.draw do
  resources :cats
  get 'cats/index'

  # For details on the DSL available within this file, see http://guides.rubyon
  root 'cats#index'
end
```

4. Save the file and close it.
5. Test the Rails app as instructed before (#test-locally).

Using Cloud SQL for PostgreSQL with App Engine

Cloud SQL for PostgreSQL is a fully-managed database service to set up, maintain, manage, and administer your relational PostgreSQL databases in Google Cloud. You can use Cloud SQL in a Rails app like any other relational database.

Set up Cloud SQL for PostgreSQL

To begin using Cloud SQL with your Rails app in production:

1. Add the `pg` and `appengine` gems to the `Gemfile` file.

```
bundle add pg
bundle add appengine
```

The Rails `Gemfile` contains the following additional `gem` entries:

appengine/rails-cloudsql-postgres/Gemfile (https://github.com/GoogleCloudPlatform/ruby-docs-samples/blob/0a6e898a1acd140ca04c63c7b68da1468acb4082/appengine/rails-cloudsql-postgres/Gemfile)

A6E898A1ACD140CA04C63C7B68DA1468ACB4082/APPENGINE/RAILS-CLOUDSQL-POSTGRES/GEMFILE)

```
# Added at 2017-08-23 15:24:52 -0700 by franknatividad:
gem "pg", "~> 0.21.0"
```

```
# Added at 2017-08-07 11:54:12 -0700 by USER:
gem "appengine", "~> 0.4.1"
```

★ **Note:** The **`appengine`** gem provides a Rake task to run database migrations in production.

2. To configure the Rails app to connect with Cloud SQL, open the **`config/database.yml`** file. The following boilerplate database settings for SQLite are displayed:

appengine/rails-cloudsql-postgres/config/boilerplate.database.yml (https://github.com/GoogleCloudPlatform/ruby-docs-samples/blob/0a6e898a1acd140ca04c63c7b68da1468acb4082/appengine/rails-cloudsql-postgres/config/boilerplate.database.yml)

368DA1468ACB4082/APPENGINE/RAILS-CLOUDSQL-POSTGRES/CONFIG/BOILERPLATE.DATABASE.YML)

```
# SQLite version 3.x
#   gem install sqlite3
#
#   Ensure the SQLite 3 gem is defined in your Gemfile
#   gem 'sqlite3'
#
default: &default
  adapter: sqlite3
  pool: 5
  timeout: 5000

development:
  <<: *default
  database: db/development.sqlite3

# Warning: The database defined as "test" will be erased and
# re-generated from your development database when you run "rake".
# Do not set this db to the same as development or production.
test:
  <<: *default
  database: db/test.sqlite3

production:
  <<: *default
  database: db/production.sqlite3
```

3. Configure the Cloud SQL instance connection name for the App Engine production environment.

a. Retrieve the instance connection name.

```
gcloud sql instances describe rails-cloudsql-instance
```

b. Copy the value next to `connectionName`.

4. Modify the `database.yml` production database configuration to the following:

appengine/rails-cloudsql-postgres/config/database.yml (https://github.com/GoogleCloudPlatform/ruby-docs-samples/blob/0a6e898a1acd140ca04c63c7b68da1468acb4082/appengine/rails-cloudsql-postgres/config/database.yml)

40CA04C63C7B68DA1468ACB4082/APPENGINE/RAILS-CLOUDSQL-POSTGRES/CONFIG/DATABASE.YML)

```
production:
  adapter: postgresql
  encoding: unicode
  pool: 5
  timeout: 5000
  username: "[YOUR_POSTGRES_USERNAME]"
  password: "[YOUR_POSTGRES_PASSWORD]"
  database: "cat_list_production"
  host:   "/cloudsql/[YOUR_INSTANCE_CONNECTION_NAME]"
```

Where:

- `[YOUR_POSTGRES_USERNAME]` represents your PostgreSQL username.
- `[YOUR_POSTGRES_PASSWORD]` represents your PostgreSQL password.
- `[YOUR_INSTANCE_CONNECTION_NAME]` represents the instance connection name that you copied in the previous step.

The Rails app is now set up to use Cloud SQL when deploying to App Engine flexible environment.

Deploying the app to the App Engine flexible environment

App Engine flexible environment uses a file called app.yaml (https://cloud.google.com/appengine/docs/flexible/ruby/configuring-your-app-with-app-yaml) to describe an app's deployment configuration. If this file isn't present, the Cloud SDK tries to guess the

deployment configuration. However, you should define the file to provide required configuration settings for the Rails secret key and Cloud SQL.

To configure the sample app for deployment to App Engine, create a new file named `app.yaml` at the root of the Rails app directory and add the following:

appengine/rails-cloudsql-postgres/app.yaml (https://github.com/GoogleCloudPlatform/ruby-docs-samples/blob/0a6e898a1acd140ca04c63c7b68da1468acb4082/appengine/rails-cloudsql-postgres/app.yaml)

5E898A1ACD140CA04C63C7B68DA1468ACB4082/APPENGINE/RAILS-CLOUDSQL-POSTGRES/APP.YAML)

```
entrypoint: bundle exec rackup --port $PORT
env: flex
runtime: ruby
```

Configure the Rails secret key in the `app.yaml` file

When a Rails app is deployed to the `production` environment, set the environment variable `SECRET_KEY_BASE` with a secret key to protect user session data. This environment variable is read from the `config/secrets.yml` file in your Rails app.

1. Generate a new secret key.

   ```
   bundle exec bin/rails secret
   ```

2. Copy the generated secret key.
3. Open the `app.yaml` file that you created earlier, and add an `env_variables` section. The `env_variables` defines environment variables in the App Engine flexible environment. The `app.yaml` file should look similar to the following example with `[SECRET_KEY]` replaced with your secret key.

   appengine/rails-cloudsql-postgres/app.yaml (https://github.com/GoogleCloudPlatform/ruby-docs-samples/blob/0a6e898a1acd140ca04c63c7b68da1468acb4082/appengine/rails-cloudsql-postgres/app.yaml)

5E898A1ACD140CA04C63C7B68DA1468ACB4082/APPENGINE/RAILS-CLOUDSQL-POSTGRES/APP.YAML)

```
entrypoint: bundle exec rackup --port $PORT
env: flex
runtime: ruby

env_variables:
  SECRET_KEY_BASE: [SECRET_KEY]
```

Configure the Cloud SQL instance in the `app.yaml` file

Next, configure the App Engine flexible environment to use a specified Cloud SQL instance by providing the Cloud SQL instance connection name in the `app.yaml` configuration file.

1. Open the `app.yaml` file, and add a new section named `beta_settings`.
2. Define a nested parameter `cloud_sql_instances` with the instance connection name as the value.

★ **Note:** You can retrieve the Cloud SQL instance connection name by running `gcloud sql instances describe rails-cloudsql-instance`.

The `app.yaml` should look similar to the following:

appengine/rails-cloudsql-postgres/app.yaml (https://github.com/GoogleCloudPlatform/ruby-docs-samples/blob/0a6e898a1acd140ca04c63c7b68da1468acb4082/appengine/rails-cloudsql-postgres/app.yaml)

5E898A1ACD140CA04C63C7B68DA1468ACB4082/APPENGINE/RAILS-CLOUDSQL-POSTGRES/APP.YAML)

```
entrypoint: bundle exec rackup --port $PORT
env: flex
runtime: ruby

env_variables:
  SECRET_KEY_BASE: [SECRET_KEY]

beta_settings:
  cloud_sql_instances: [YOUR_INSTANCE_CONNECTION_NAME]
```

Create an App Engine flexible environment app

If this is the first time you are deploying an app, you need to create an App Engine flexible environment app and select the region where you want to run the Rails app.

1. Create an App Engine app.

```
gcloud app create
```

2. Select a region that supports App Engine flexible environment for Ruby apps. Read more about Regions and zones (https://cloud.google.com/docs/geography-and-regions).

Deploy a new version

Next, deploy a new version of the Rails app described in the `app.yaml` file without redirecting traffic from the current default serving version.

1. Precompile the Rails assets.

```
bundle exec bin/rails assets:precompile
```

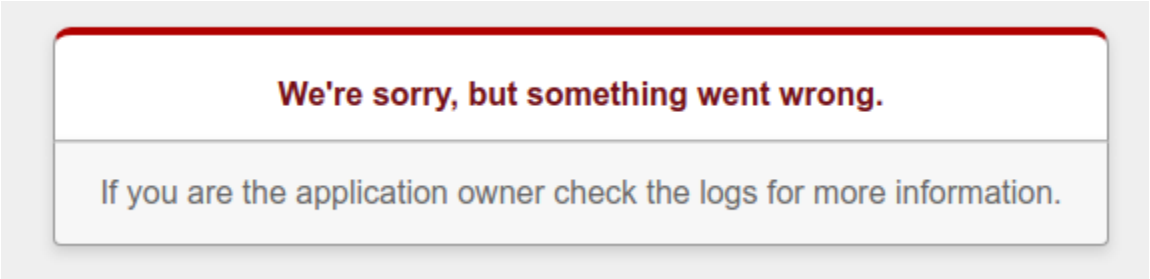
2. After the assets finish compiling, deploy a new version of the app.

```
gcloud app deploy --no-promote
```

It can take several minutes to finish the deployment. Wait for a success message. You can view deployed versions in the App Engine version list (https://console.cloud.google.com/appengine/versions).

Warning: The older versions of your app remain, as do their associated VM instances. Be aware that all of these app versions and VM instances are billable resources.

After you deploy the new version, if you attempt to access this new version, it shows the following error message because you haven't migrated the database.



Grant required permission for the `appengine` gem

Next, grant access to the cloudbuild service account to run production database migrations with the `appengine` gem.

1. List available projects.

```
gcloud projects list
```

2. In the output, find the project you want to use to deploy your app, and copy the project number.
3. Add a new member to your project IAM policy for the role `roles/editor` to run database migrations. Replace `[YOUR-PROJECT-ID]` with your Google Cloud project ID and replace`[PROJECT_NUMBER]` with the project number you copied in the previous step.

```
gcloud projects add-iam-policy-binding [YOUR-PROJECT-ID] \
  --member=serviceAccount:[PROJECT_NUMBER]@cloudbuild.gserviceaccount.com \
  --role=roles/editor
```

Migrate your Rails database

Rails database migrations are used to update the schema of your database without using SQL syntax directly. Next you migrate your `cat_list_production` database.

The `appengine` gem provides the Rake task `appengine:exec` to run a command against the most recent deployed version of your app in the production App Engine flexible environment.

1. Migrate the Cloud SQL for PostgreSQL `cat_list_production` database in production.

```
bundle exec rake appengine:exec -- bundle exec rake db:migrate
```

You should see output similar to:

```
---------- EXECUTE COMMAND ----------
bundle exec rake db:migrate
Debuggee gcp:787021104993:8dae9032f8b02004 successfully registered
== 20170804210744 CreateCats: migrating =======================================
-- create_table(:cats)
   -> 0.0219s
== 20170804210744 CreateCats: migrated (0.0220s) ==============================

---------- CLEANUP ----------
```

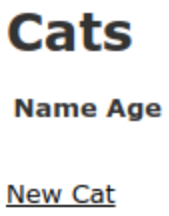
★ **Note:** If a permission error occurs when attempting to run database migrations, verify that `[PROJECT_NUMBER]@cloudbuild.gserviceaccount.com` has the `roles/editor` role.

2. To verify the database migration, go to:

```
[YOUR-VERSION]-dot-[YOUR-PROJECT-ID].appspot.com
```

★ **Note:** To get a list of versions, use `gcloud app versions list`. The last **default** service version item is the latest deployment.

The following is displayed for a successful deployment:



Migrate traffic to new version

Finally, direct traffic to your newly deployed version by using the following command:

```
gcloud app services set-traffic default --splits [YOUR-VERSION]=1
```

The new version of the app is now accessible from:

```
https://[YOUR-PROJECT-ID].appspot.com
```

Reading App Engine logs

Now that you have deployed your Rails app, you might want to read the logs. You can read the app logs by using the Logs Viewer (https://console.cloud.google.com/logs/viewer) located in the Google Cloud Console.

You can learn more about reading logs using the Cloud SDK (https://cloud.google.com/sdk/gcloud/reference/app/logs/read).

Clean up resources

After you've finished the Using Cloud SQL for PostgreSQL with Rails 5 tutorial, you can clean up the resources that you created on GCP so they won't take up quota and you won't be billed for them in the future. The following sections describe how to delete or turn off these resources.

Delete project

The easiest way to eliminate billing is to delete the project that you created for the tutorial.

To delete the project:

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 THE MANAGE RESOURCES PAGE (HTTPS://CONSOLE.CLOUD.GOOGLE.COM/IAM-ADMIN/PRC

2. In the project list, select the project you want to delete and click **Delete** .
3. In the dialog, type the project ID, and then click **Shut down** to delete the project.

Delete an App Engine version

To delete an app version:

1. In the Cloud Console, go to the **Versions** page for App Engine.

   GO TO THE VERSIONS PAGE (HTTPS://CONSOLE.CLOUD.GOOGLE.COM/APPENGINE/VERSIONS)

2. Select the checkbox for the non-default app version you want to delete.

★ **Note:** The only way you can delete the default version of your App Engine app is by deleting your project. However, you can stop the default version in the Cloud Console (https://console.cloud.google.com/appengine/versions). This action shuts down all instances associated with the version. You can restart these instances later if needed.

In the App Engine standard environment, you can stop the default version only if your app has manual or basic scaling.

3. Click **Delete** to delete the app version.

Delete a Cloud SQL instance

To delete a Cloud SQL instance:

1. In the Cloud Console, go to the **SQL Instances** page.

   GO TO THE SQL INSTANCES PAGE (HTTPS://CONSOLE.CLOUD.GOOGLE.COM/SQL/INSTANCES)

2. Click the name of the SQL instance you want to delete.
3. Click **Delete** to delete the instance.

What's next

- Learn how to run the Ruby Bookshelf sample app in the App Engine flexible environment (https://cloud.google.com/ruby/getting-started/tutorial-app).
- Learn how to run the Ruby Bookshelf sample app on Compute Engine (https://cloud.google.com/ruby/tutorials/bookshelf-on-compute-engine).
- Learn how to run the Ruby Bookshelf sample app on Google Kubernetes Engine (https://cloud.google.com/ruby/tutorials/bookshelf-on-kubernetes-engine).

Last updated December 4, 2019.