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

Using PostgreSQL with Ruby

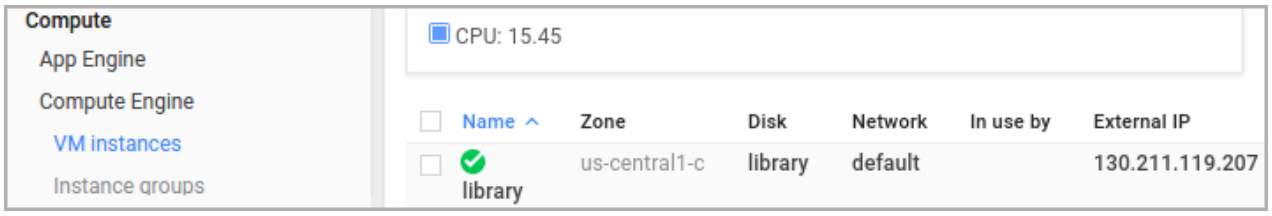
This part of the Bookshelf app tutorial shows how the sample app stores its persistent data in a PostgreSQL database.

This sample uses a PostgreSQL server, running on a Compute Engine virtual machine instance, to store its persistent data. If you prefer to use a different PostgreSQL server, you can deploy this sample app to Google Cloud and configure it to use any PostgreSQL server of your choice.

This page is part of a multipage tutorial. To start from the beginning and read the setup instructions, go to Ruby Bookshelf app (https://cloud.google.com/ruby/getting-started/tutorial-app).

Running PostgreSQL on Compute Engine

1. Use the PostgreSQL image (https://cloud.google.com/marketplace/solution/bitnami-launchpad/postgresql) provided by Bitnami in the Google Cloud Marketplace.
2. Click **Launch on Compute Engine**. It takes a few minutes to create the instance and deploy PostgreSQL.
3. When the deployment is done, make a note of the **Admin user** and **Admin password** for your PostgreSQL database.
4. Click the instance link to go to the VM instance details. Make a note of the **External IP** address of your instance.



5. To edit the instance, click **Edit** . In the **Network tags** field, enter `postgres-bitnami`, and then click **Save**.
6. Create a firewall rule to allow traffic to PostgreSQL on instances with the `postgres-bitnami` network tag.

```
gcloud compute firewall-rules create default-allow-postgresql \
    --allow tcp:5432 \
    --source-ranges 0.0.0.0/0 \
    --target-tags postgres-bitnami \
    --description "Allow access to PostgreSQL from all IPs"
```

Warning: This configuration allows anyone with the password to log in to your PostgreSQL instance from anywhere. It is only for demonstration purposes. In production environments, create secure firewall rules and configure PostgreSQL security.

Configuring settings

1. Go to the `getting-started-ruby/2-postgresql` directory, and copy the sample `database.yml` file:

```
cp config/database.example.yml config/database.yml
```

2. To configure your database, edit the `config/database.yml` file. Set the value of `database` to `bookshelf`. Replace the `[YOUR_POSTGRES_*]` placeholders with the specific values for your PostgreSQL instance and database.

   For example, suppose your IPv4 address is `173.194.230.44`, your username is `postgres`, and your password is `secret123`. Then the `postgresql_settings` section of your `database.yml` file would look like this:

```
postgresql_settings: &postgresql_settings
  adapter: postgresql
```

```
encoding: unicode
pool: 5
username: postgres
password: secret123
host: 173.194.230.44
database: bookshelf
```

Installing dependencies

In the **`2-postgresql`** directory enter the following command:

```
bundle install
```

Creating a database and tables

1. Create the database and the required tables.

```
bundle exec rake db:create
bundle exec rake db:migrate
```

Running the app on your local machine

1. Start a local web server.

```
bundle exec rails server
```

2. In your web browser, enter the following address:

   http://localhost:3000 (http://localhost:3000)

Now you can browse the app's web pages to add, edit, and delete books.

To exit the local web server, press `Control+C`.

Deploying the app to the App Engine flexible environment

1. Compile the JavaScript assets for production.

```
RAILS_ENV=production bundle exec rake assets:precompile
```

2. Deploy the sample app.

```
gcloud app deploy
```

3. In your web browser, enter the following address.

```
https://[YOUR_PROJECT_ID].appspot.com
```

If you update your app, you can deploy the updated version by entering the same command you used to deploy the app the first time. The new deployment creates a new version (https://console.cloud.google.com/appengine/versions) of your app and promotes it to the default version. 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.

You can reduce costs by deleting the non-default versions of your app.

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.

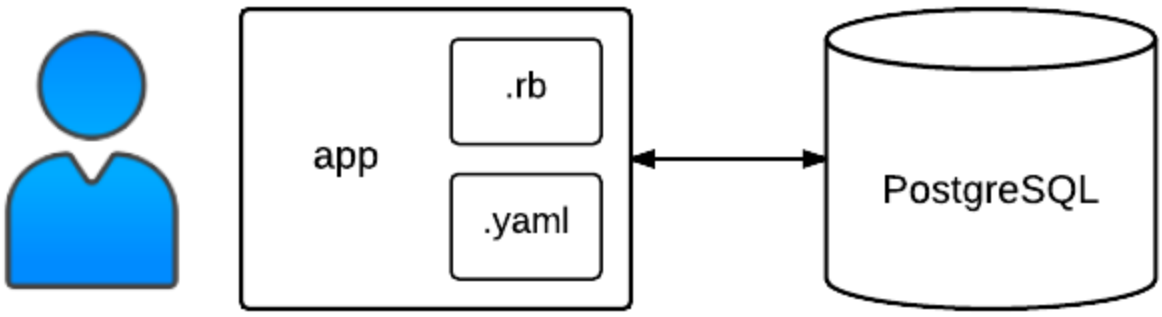
3. Click **Delete** to delete the app version.

For complete information about cleaning up billable resources, see the Cleaning up (https://cloud.google.com/ruby/getting-started/using-pub-sub#clean-up) section in the final step of

this tutorial.

App structure

The following diagram shows the app's components and how they connect to each other.



Understanding the code

This section walks you through the app's code and explains how it works.

List books

When you visit the app's home page, you are routed to the `index` action of the `BooksController` class. This is configured in the `config/routes.rb` file.

2-postgresql/config/routes.rb (https://github.com/GoogleCloudPlatform/getting-started-ruby/blob/steps/2-postgresql/config/routes.rb)

GOOGLECLOUDPLATFORM/GETTING-STARTED-RUBY/BLOB/STEPS/2-POSTGRESQL/CONFIG/ROUTES.RB)

```
Rails.application.routes.draw do

  # Route root of application to BooksController#index action
  root "books#index"

  # Restful routes for BooksController
  resources :books

end
```

The `BookController#index` action retrieves a list of books from the Cloud SQL database. The app lists at most 10 books on each web page, so the list depends on which page the user is viewing. For example, suppose there are 26 books in the database, and the user is on the third page (`/?page=3`). In that case, `params[:page]` is equal to 3, which is assigned to the `page_number` variable. Then a list of 6 books, starting at offset 20, is retrieved and assigned to `@books`.

2-postgresql/app/controllers/books_controller.rb (https://github.com/GoogleCloudPlatform/getting-started-ruby/blob/steps/2-postgresql/app/controllers/books_controller.rb)

/GETTING-STARTED-RUBY/BLOB/STEPS/2-POSTGRESQL/APP/CONTROLLERS/BOOKS_CONTROLLER.RB)

```
class BooksController < ApplicationController

  PER_PAGE = 10

  def index
    page_number = params[:page] ? params[:page].to_i : 1
    book_offset = PER_PAGE * (page_number - 1)
    @books      = Book.limit(PER_PAGE).offset(book_offset)
    @next_page  = page_number + 1 if @books.count == PER_PAGE
  end
```

The `Book` class is a simple ActiveRecord (http://guides.rubyonrails.org/active_record_basics.html) model that represents an individual book in the books table.

2-postgresql/app/models/book.rb (https://github.com/GoogleCloudPlatform/getting-started-ruby/blob/steps/2-postgresql/app/models/book.rb)

GLECLOUDPLATFORM/GETTING-STARTED-RUBY/BLOB/STEPS/2-POSTGRESQL/APP/MODELS/BOOK.RB)

```
class Book < ActiveRecord::Base
  validates :title, presence: true
end
```

In the `routes.rb` file, the `resources :books` call configures RESTful routes for creating, reading, updating, and deleting books that are routed to the corresponding actions in the `BooksController` class.

After `BooksController.index` retrieves a list of books, the embedded Ruby code in the `books/index.html.erb` file renders the list.

2-postgresql/app/views/books/index.html.erb (https://github.com/GoogleCloudPlatform/getting-started-ruby/blob/steps/2-postgresql/app/views/books/index.html.erb)

ATFORM/GETTING-STARTED-RUBY/BLOB/STEPS/2-POSTGRESQL/APP/VIEWS/BOOKS/INDEX.HTML.ERB)

```
<% @books.each do |book| %>
  <div class="media">
    <%= link_to book_path(book) do %>
      <div class="media-body">
        <h4><%= book.title %></h4>
        <p><%= book.author %></p>
      </div>
    <% end %>
  </div>
<% end %>

<% if @next_page %>
  <nav>
    <ul class="pager">
      <li><%= link_to "More", books_path(page: @next_page) %></li>
    </ul>
  </nav>
<% end %>
```

Display book details

When you click an individual book on the web page, the `BookController#show` action retrieves the book, specified by its ID, from the books table.

2-postgresql/app/controllers/books_controller.rb (https://github.com/GoogleCloudPlatform/getting-started-ruby/blob/steps/2-postgresql/app/controllers/books_controller.rb)

/GETTING-STARTED-RUBY/BLOB/STEPS/2-POSTGRESQL/APP/CONTROLLERS/BOOKS_CONTROLLER.RB)

```
def show
  @book = Book.find params[:id]
end
```

Then the embedded Ruby code in the `show.html.erb` file displays the book's details.

2-postgresql/app/views/books/show.html.erb (https://github.com/GoogleCloudPlatform/getting-started-ruby/blob/steps/2-postgresql/app/views/books/show.html.erb)

ATFORM/GETTING-STARTED-RUBY/BLOB/STEPS/2-POSTGRESQL/APP/VIEWS/BOOKS/SHOW.HTML.ERB)

```
<div class="media">
  <div class="media-body">
    <h4><%= @book.title %> |   <small><%= @book.published_on %></small></h4>
    <h5>By <%= @book.author || "unknown" %></h5>
    <p><%= @book.description %></p>
  </div>
</div>
```

Create books

When you click **Add book** on the web page, the `BooksController#new` action creates a new book. The embedded Ruby code in the `new.html.erb` file points to `_form.html.erb`, which displays the form for adding a new book.

2-postgresql/app/views/books/_form.html.erb (https://github.com/GoogleCloudPlatform/getting-started-ruby/blob/steps/2-postgresql/app/views/books/_form.html.erb)

ıTFORM/GETTING-STARTED-RUBY/BLOB/STEPS/2-POSTGRESQL/APP/VIEWS/BOOKS/_FORM.HTML.ERB)

```
<%= form_for @book do |f| %>
  <div class="form-group">
    <%= f.label :title %>
    <%= f.text_field :title %>
  </div>
  <div class="form-group">
    <%= f.label :author %>
    <%= f.text_field :author %>
  </div>
  <div class="form-group">
    <%= f.label :published_on, "Date Published" %>
    <%= f.date_field :published_on %>
  </div>
  <div class="form-group">
    <%= f.label :description %>
```

```
    <%= f.text_area :description %>
  </div>
  <button class="btn btn-success" type="submit">Save</button>
<% end %>
```

When you submit the form, the `BooksController#create` action saves the book in the database. If the new book is saved successfully, the book's page is displayed. Otherwise, the form is displayed again along with error messages. The `book_params` method uses strong parameters (http://guides.rubyonrails.org/action_controller_overview.html#strong-parameters) to specify which form fields are allowed. In this case, only book title, author, publication date, and description are allowed.

2-postgresql/app/controllers/books_controller.rb (https://github.com/GoogleCloudPlatform/getting-started-ruby/blob/steps/2-postgresql/app/controllers/books_controller.rb)

/GETTING-STARTED-RUBY/BLOB/STEPS/2-POSTGRESQL/APP/CONTROLLERS/BOOKS_CONTROLLER.RB)

```
def create
  @book = Book.new book_params

  if @book.save
    flash[:success] = "Added Book"
    redirect_to book_path(@book)
  else
    render :new
  end
end

private

def book_params
  params.require(:book).permit(:title, :author, :published_on, :description)
end
```

Edit books

When you click **Edit book** on the web page, the `BooksController#update` action retrieves the book from the database. The embedded Ruby code in the `edit.html.erb` file points to `_form.html.erb`, which displays the form for editing the book.

2-postgresql/app/controllers/books_controller.rb (https://github.com/GoogleCloudPlatform/getting-started-ruby/blob/steps/2-postgresql/app/controllers/books_controller.rb)

/GETTING-STARTED-RUBY/BLOB/STEPS/2-POSTGRESQL/APP/CONTROLLERS/BOOKS_CONTROLLER.RB)

```
def update
  @book = Book.find params[:id]

  if @book.update book_params
    flash[:success] = "Updated Book"
    redirect_to book_path(@book)
  else
    render :edit
  end
end
```

When you submit the form, the `BooksController#update` action saves the book in the database. If the new book is saved successfully, the book's page is displayed. Otherwise, the form is displayed again along with error messages.

Delete books

When you click **Delete Book** on the web page, the `BooksController#destroy` action deletes the book from the database and then displays the list of books.

2-postgresql/app/controllers/books_controller.rb (https://github.com/GoogleCloudPlatform/getting-started-ruby/blob/steps/2-postgresql/app/controllers/books_controller.rb)

/GETTING-STARTED-RUBY/BLOB/STEPS/2-POSTGRESQL/APP/CONTROLLERS/BOOKS_CONTROLLER.RB)

```
def destroy
  @book = Book.find params[:id]
  @book.destroy
  redirect_to books_path
end
```

< PREV (HTTPS://CLOUD.GOOGLE.COM/RUBY/GETTING-STARTED/USING-STRUCTURED-DATA)

NEXT > (HTTPS://CLOUD.GOOGLE.COM/RUBY/GETTING-STARTED/USING-CLOUD-STORAGE)

Last updated November 19, 2019.