Using Ruby on Rails with Cloud SQL for PostgreSQL

GSP109

Google Cloud Self-Paced Labs

Overview

In this Qwiklab, you will learn how to deploy a new Ruby on Rails application using Google Cloud SQL for PostgreSQL to Google App Engine Flexible environment.

Google Cloud SQL for PostgreSQL is a fully-managed database service that makes it easy to set up, maintain, manage, and administer your PostgreSQL relational databases on Google Cloud.

Google App Engine Flexible environment applications are easy to create, maintain, and scale as your traffic and data storage changes. With App Engine, there are no servers to maintain. You simply upload your application and it's ready to go.

App Engine applications automatically scale based on incoming traffic. Load balancing, microservices, authorization, SQL and NoSQL databases, traffic splitting, logging, search, versioning, roll out and roll backs, and security scanning are all supported natively and are highly customizable.

What you'll learn

- How to enable the Cloud SQL Admin API
- How to create a PostgreSQL Cloud SQL instance
- How to set up the Cloud SQL Proxy
- How to set up a Rails app with PostgreSQL
- How to enable the App Engine Admin API
- How to deploy a Rails app to App Engine flexible environment

What you'll need

- A Browser, such Chrome or Firefox
- Familiarity with standard Linux text editors such as Vim, Emacs or Nano
- Familiarity using Ruby

Setup and Requirements

Qwiklabs setup

Before you click the Start Lab button

Read these instructions. Labs are timed and you cannot pause them. The timer, which starts when you click **Start Lab**, shows how long Google Cloud resources will be made available to you.

This Qwiklabs hands-on lab lets you do the lab activities yourself in a real cloud environment, not in a simulation or demo environment. It does so by giving you new, temporary credentials that you use to sign in and access Google Cloud for the duration of the lab.

What you need

To complete this lab, you need:

- Access to a standard internet browser (Chrome browser recommended).
- Time to complete the lab.
  Note: If you already have your own personal Google Cloud account or project, do not use it for this lab.

Note: If you are using a Pixelbook, open an Incognito window to run this lab.

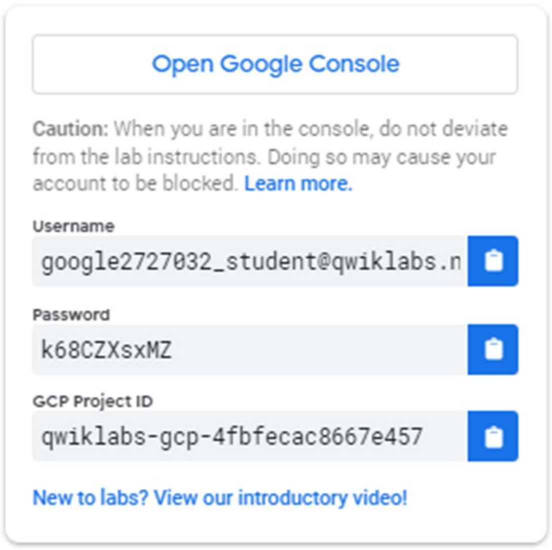
Ruby VM

Installing Ruby on Rails and building a default app to work with can take a decent amount of time. This lab aims to cut down that time by using a virtual machine with Ruby on Rails already installed and a prebuilt app ready to test and edit. As soon as you start the lab, this `ruby-vm` machine will be provisioned and, throughout the lab, you will be connecting to it using it as your workspace.

Cloud Console

How to start your lab and sign in to the Google Cloud Console

1. Click the **Start Lab** button. If you need to pay for the lab, a pop-up opens for you to select your payment method. On the left is a panel populated with the temporary credentials that you must use for this lab.



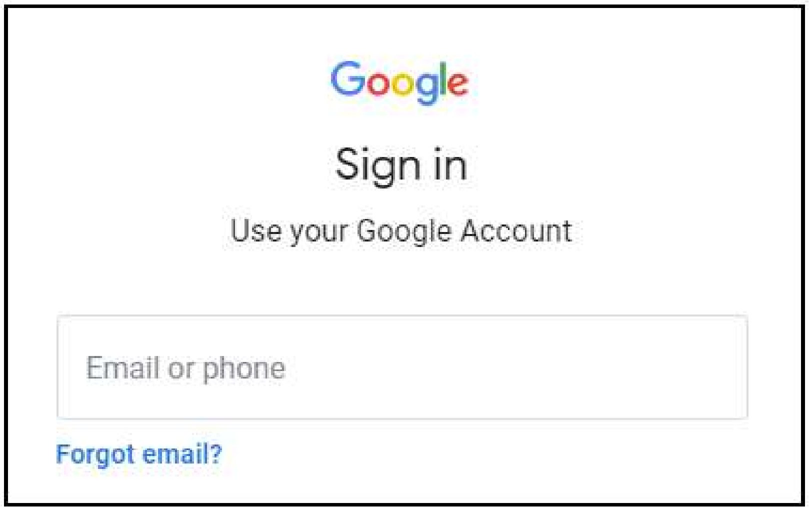
Caution: When you are in the console, do not deviate from the lab instructions. Doing so may cause your account to be blocked. Learn more.

Username
google2727032_student@qwiklabs.n

Password
k68CZXsxMZ

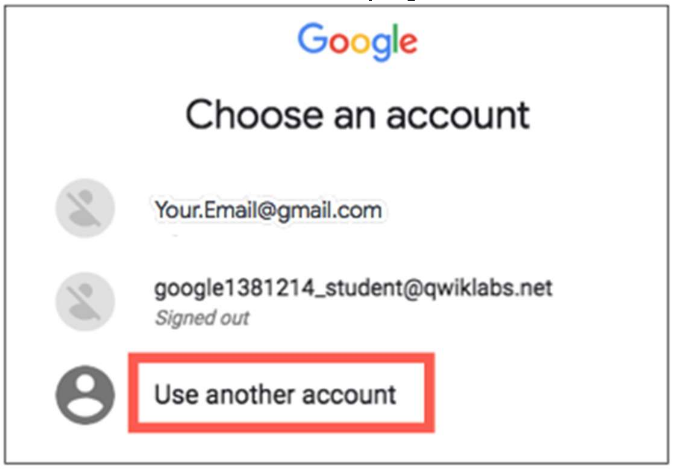
GCP Project ID
qwiklabs-gcp-4fbfecac8667e457

2. Copy the username, and then click **Open Google Console**. The lab spins up resources, and then opens another tab that shows the **Sign in** page.



Tip: Open the tabs in separate windows, side-by-side.

If you see the **Choose an account** page, click **Use Another Account**.



3. In the **Sign in** page, paste the username that you copied from the Connection Details panel. Then copy and paste the password.

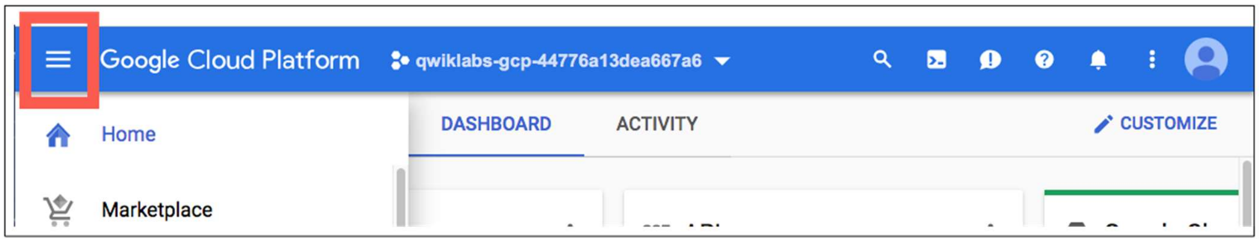
   Important: You must use the credentials from the Connection Details panel. Do not use your Qwiklabs credentials. If you have your own Google Cloud account, do not use it for this lab (avoids incurring charges).

4. Click through the subsequent pages:

   - Accept the terms and conditions.
   - Do not add recovery options or two-factor authentication (because this is a temporary account).
   - Do not sign up for free trials.

After a few moments, the Cloud Console opens in this tab.

Note: You can view the menu with a list of Google Cloud Products and Services by clicking the **Navigation menu** at the top-
left.

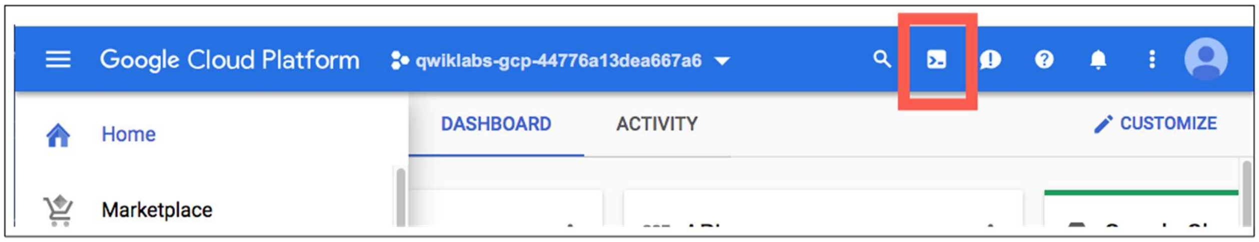


Cloud Shell

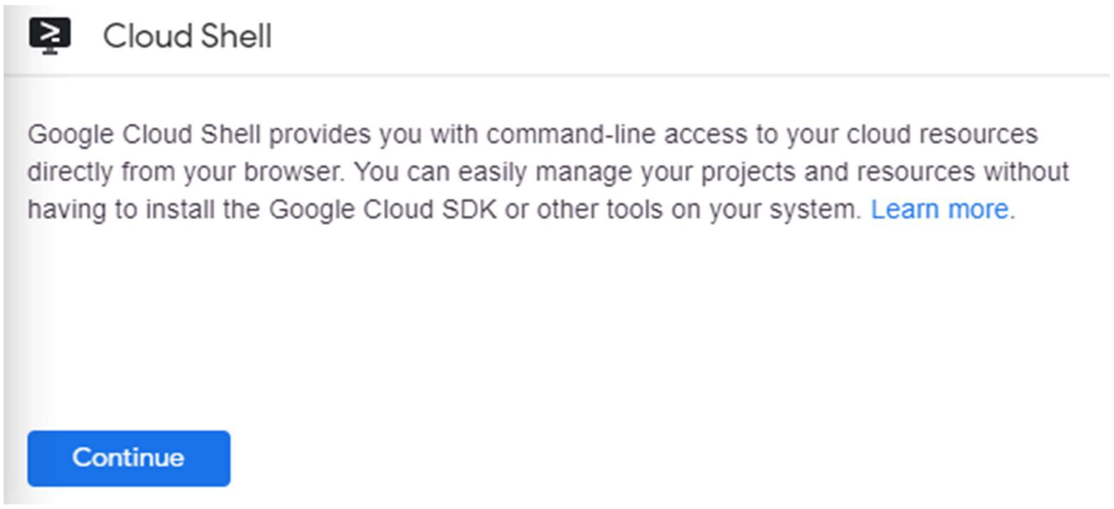
Activate Cloud Shell

Cloud Shell is a virtual machine that is loaded with development tools. It offers a persistent 5GB home directory and runs on the Google Cloud. Cloud Shell provides command-line access to your Google Cloud resources.

In the Cloud Console, in the top right toolbar, click the **Activate Cloud Shell** button.

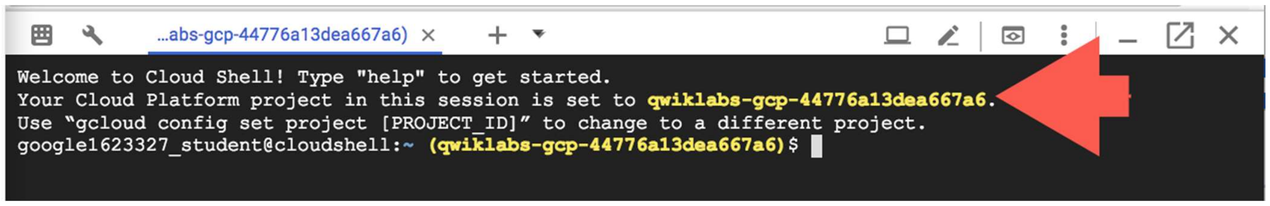


Click **Continue**.


Cloud Shell

Google Cloud Shell provides you with command-line access to your cloud resources directly from your browser. You can easily manage your projects and resources without having to install the Google Cloud SDK or other tools on your system. Learn more.

It takes a few moments to provision and connect to the environment. When you are connected, you are already authenticated, and the project is set to your *PROJECT_ID*. For example:



```
Welcome to Cloud Shell! Type "help" to get started.
Your Cloud Platform project in this session is set to qwiklabs-gcp-44776a13dea667a6.
Use "gcloud config set project [PROJECT_ID]" to change to a different project.
google1623327_student@cloudshell:~ (qwiklabs-gcp-44776a13dea667a6)$
```

`gcloud` is the command-line tool for Google Cloud. It comes pre-installed on Cloud Shell and supports tab-completion.

You can list the active account name with this command:

```
gcloud auth list
```

(Output)

```
Credentialed accounts:
 - <myaccount>@<mydomain>.com (active)
```

(Example output)

```
Credentialed accounts:
 - google1623327_student@qwiklabs.net
```

You can list the project ID with this command:

```
gcloud config list project
```

(Output)

```
[core]
project = <project_ID>
```

(Example output)

```
[core]
project = qwiklabs-gcp-44776a13dea667a6
```

For full documentation of `gcloud` see the gcloud command-line tool overview.

Code Editor

In this lab you will add and edit files. You can use your choice of `vim`, `nano`, or `emacs`.

Create a PostgreSQL Cloud SQL instance

1. First, in Cloud Shell, ssh into your `ruby-vm`:

```
gcloud compute ssh --zone us-central1-a ruby-vm
```

Type and enter **y** when prompted and then press enter twice to use a blank password.

2. Now in the VM, create a PostgreSQL instance named `postgres-instance`that you will use later in the lab:

```
gcloud sql instances create postgres-instance \
    --database-version POSTGRES_9_6 \
    --tier db-g1-small
```

Creating a new PostgreSQL instance will take a few minutes.

Test Completed Task

Click **Check my progress** to verify your performed task. If you have completed the task successfully you will granted with an assessment score.

Create a PostgreSQL Cloud SQL instance

Check my progress

3. Set the password for the **postgres** user. Run the following, replacing `[PASSWORD]` with a password of your choice:

```
gcloud sql users set-password postgres --host=% \
    --instance postgres-instance \
    --password [PASSWORD]
```

The PostgreSQL instance is now ready.

Set up the Cloud SQL Proxy

When deployed, your application uses the Cloud SQL Proxy built into the App Engine environment to communicate with your Cloud SQL instance. However, to test your application using the Cloud Shell, you must install a copy of the Cloud SQL Proxy in the development environment.

In the next steps, you'll download and set up the Cloud SQL Proxy using Cloud Shell.

Download the Cloud SQL Proxy:

```
wget https://dl.google.com/cloudsql/cloud_sql_proxy.linux.amd64
```

Rename the proxy to use the standard filename:

```
mv cloud_sql_proxy.linux.amd64 cloud_sql_proxy
```

Make the proxy executable:

```
chmod +x cloud_sql_proxy
```

Create a directory where the proxy sockets will live:

```
sudo mkdir /cloudsql

sudo chmod 0777 /cloudsql
```

The proxy is now ready to be used. Next, you will obtain the instance connection name for the Cloud SQL for PostgreSQL instance and use it to connect the `cloud_sql_proxy` to the instance you created in the previous step. The instance connection name is `connectionName.`

Retrieve the instance connection name by running:

```
gcloud sql instances describe postgres-instance | grep connectionName
```

Tip: Copy the `connectionName` output to your computer, you'll be using it again during this lab.

Run the following to start the Cloud SQL Proxy, and replace `[YOUR_INSTANCE_CONNECTION_NAME]` with the connectionName value:

```
./cloud_sql_proxy -dir=/cloudsql \
                  -instances="[YOUR_INSTANCE_CONNECTION_NAME]"
```

You should see the following output:

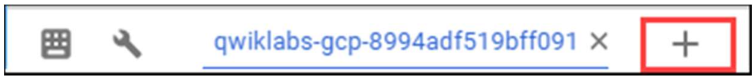
```
2017/04/19 12:42:21 Listening on /cloudsql/postgres-rails:us-central1:postgres-
instance/.s.PGSQL.5432 for postgres-rails:us-central1:postgres-instance
2017/04/19 12:42:21 Ready for new connections
```

Keep this Cloud Shell instance open to run the Cloud SQL Proxy.

Open a separate Cloud Shell tab

While the Cloud SQL Proxy is running, open a new Cloud Shell instance to run the remaining steps for this lab.

Open another Cloud Shell instance by using the **+** button to create a new tab:



Access Your Ruby App

1. First, ssh into your `ruby-vm`:

```
gcloud compute ssh --zone us-central1-a ruby-vm
```

This time there shouldn't be any extra prompts.

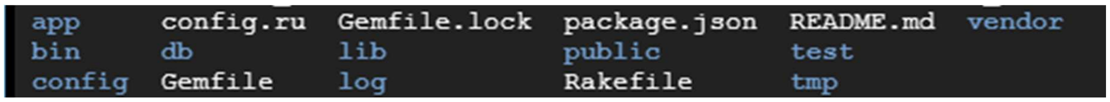
2. Next, change into the prebuilt Ruby On Rails app's directory:

```
cd /home/ruby_workspace/app_name
```

3. View the generated files for a new Rails applications:

```
ls
```

You should see something similar to:



```
app     config.ru  Gemfile.lock  package.json  README.md  vendor
bin     db         lib           public        test
config  Gemfile    log           Rakefile      tmp
```

Test the Generated Rails Application

Now that we have a Rails app, let's test it and preview it through our `ruby-vm's` external IP.

1. Run this command to list your compute engine instances:

```
gcloud compute instances list
```

You should see your `ruby-vm`. Take note of its `EXTERNAL_IP` as you will use that shortly.

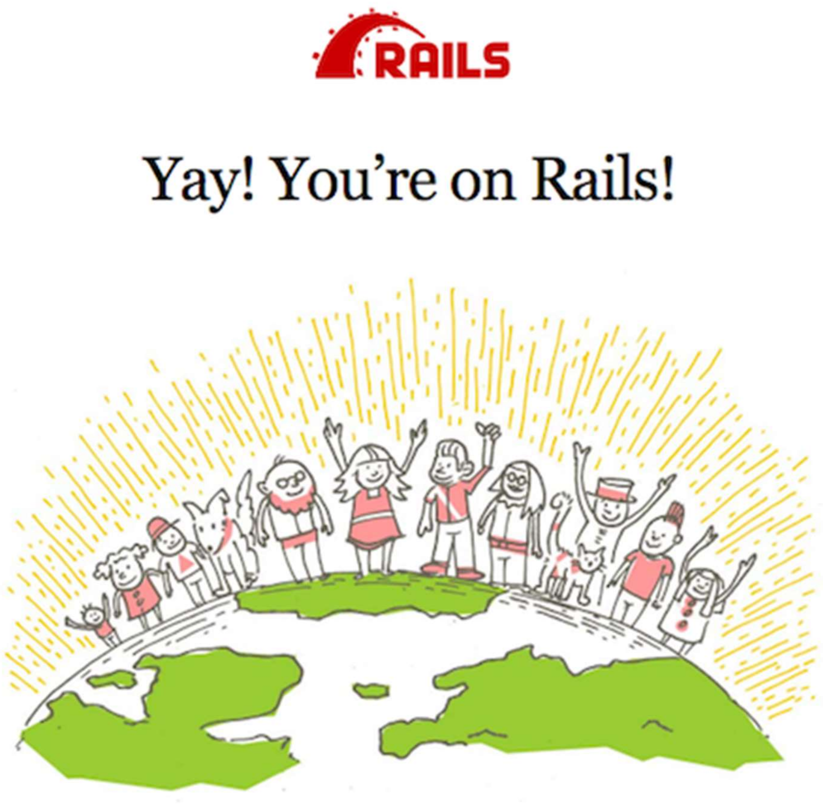
2. Start the Rails app server listening on port 8080:

```
bundle exec rails server --port 8080 -b 0.0.0.0
```

2. Once the application starts, open a new tab in your browser and enter this as the url replacing `[EXTERNAL_IP]` with your `ruby_vm's` external IP:

```
[EXTERNAL_IP]:8080
```

You should see the following image:



3. Before going to the next section, stop the Rails server by using **Ctrl** + **c** in the Cloud Shell.

Set up Rails app with Cloud SQL for PostgreSQL

You must add the **pg** gem to the file Gemfile to begin using Google Cloud SQL for PostgreSQL.

1. Add **pg** gem in Gemfile using:

```
bundle add pg
```

Open the file Gemfile. Gemfile should look something similar to the following:

Note: The file was truncated for this lab.

```
...
group :test do
  # Adds support for Capybara system testing and selenium driver
  gem 'capybara', '>= 2.15'
  gem 'selenium-webdriver'
  # Easy installation and use of chromedriver to run system tests with Chrome
  gem 'chromedriver-helper'
end

# Windows does not include zoneinfo files, so bundle the tzinfo-data gem
gem 'tzinfo-data', platforms: [:mingw, :mswin, :x64_mingw, :jruby]

gem "pg", "~> 1.1"
```

2. Next, update the installed dependencies:

```
bundle install
```

3. Configure your Rails app to communicate with Cloud SQL for PostgreSQL. Open the file `config/database.yml`, you should see the following boilerplate database settings for SQLite:

```
# 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
```

4. Modify the `database.yml` production database configuration by adding the following and replacing `[PASSWORD]` and `[YOUR_INSTANCE_CONNECTION_NAME]` with their associated values:

```
production:
  adapter: postgresql
  pool: 5
  timeout: 5000
  username: postgres
  password: [PASSWORD]
  database: postgres-database
  host: /cloudsql/[YOUR_INSTANCE_CONNECTION_NAME]
```

The Rails app is now set up to use Cloud SQL for PostgreSQL in a Production environment. These instructions can also be used to set up Cloud SQL for Test and Development environments.

Generate a model

1. The new model contains a list of cat names and ages. You can generate a new Rails ActiveRecord model using the following command:

```
bundle exec rails generate model Cat name:string age:decimal
```

This generates a model named `Cat`, a database table named `cats`, and a migration file with a similar name to `db/migrate/20170302062657_create_cats.rb`.

2. Update the production Cloud SQL for PostgreSQL instance database `postgres_instance` with the following command:

```
RAILS_ENV=production bundle exec rails db:create

RAILS_ENV=production bundle exec rails db:migrate
```

You should see a similar output to the following for a successful migration.

```
== 20170302062657 CreateCats: migrating ============================
-- create_table(:cats)
   -> 0.0585s
== 20170302062657 CreateCats: migrated (0.0586s) ===================
```

Add entries

Now you'll add a few cats using the Rails console to try out the new `Cat` model.

Use the following command to start the Rails console:

```
RAILS_ENV=production bundle exec rails console
```

With the Rails console open, use the following Ruby code to add two cat friends:

```
Cat.create name: "Mr. Whiskers", age: 4

Cat.create name: "Ms. Paws", age: 2
```

Each line creates a new entry in the database containing the provided cats.

Exit the Rails console:

```
exit
```

List the different cats

Now add a page to display a list of cats to the generated Rails application.

Generate scaffolding for a new page with the `rails generate` command.

The following command creates a new Rails controller named `CatFriendsController` with an index action:

```
bundle exec rails generate controller CatFriends index
```

This generates the following new directories and files by default for the new `CatFriendsController` controller and index action:

```
create  app/controllers/cat_friends_controller.rb
 route  get 'cat_friends/index'
invoke  erb
create    app/views/cat_friends
create    app/views/cat_friends/index.html.erb
invoke  test_unit
create    test/controllers/cat_friends_controller_test.rb
invoke  helper
create    app/helpers/cat_friends_helper.rb
invoke    test_unit
invoke  assets
invoke    coffee
create      app/assets/javascripts/cat_friends.coffee
invoke    scss
create      app/assets/stylesheets/cat_friends.scss
```

Set the index controller action as the root action for Rails, and whenever a user visits the Rails app, they see the list of cats.

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

```
Rails.application.routes.draw do
  get 'cat_friends/index'
  # For details on the DSL available within this file, see
http://guides.rubyonrails.org/routing.html

end
```

Modify this file by adding `root cat_friends#index`as shown below:

```
Rails.application.routes.draw do
  get 'cat_friends/index'
  # For details on the DSL available within this file, see
http://guides.rubyonrails.org/routing.html

  root 'cat_friends#index'
end
```

Save and close the file.

Display database entries

Modify the Rails app to display entries in the database.

Open the file `app/controllers/cat_friends_controller.rb` and set an instance variable `@cats` that contains all cat entries. Your file should be similar to the following:

```
class CatFriendsController < ApplicationController
  def index
    @cats = Cat.all
  end
end
```

Next, open the file `app/views/cat_friends/index.html.erb`, and add Ruby code to display each cat entry in `@cats`. Your file should be similar to the following:

```
<h1>A list of my Cats</h1>

<% @cats.each do |cat| %>
<%=cat.name%> is <%=cat.age%> years old!<br />
<% end %>
```

Save the file and close it.

Before you can test the Rails app, you need to define a secret key for the production environment in Cloud Shell. A secret key is used to protect user session data.

Generate a secret key:

```
bundle exec rails secret
```

Save your secret key to your computer, you'll be using a couple of times in this lab.

Set the environment variable `SECRET_KEY_BASE` with your secret key as its value:

```
export SECRET_KEY_BASE=[SECRET_KEY]
```

Test the Rails app:

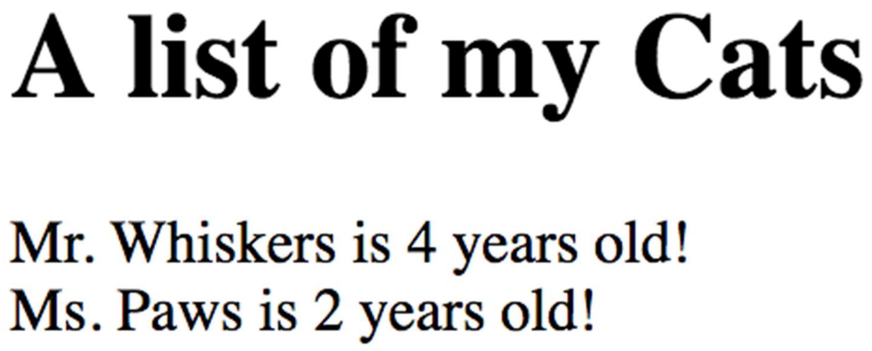
```
RAILS_ENV=production bundle exec rails assets:precompile

RAILS_ENV=production bundle exec rails server --port 8080 -b 0.0.0.0
```

Once the application starts, open a new tab in your browser and enter this as the url replacing `[EXTERNAL_IP]` with your `ruby_vm's` external IP:

```
[EXTERNAL_IP]:8080
```

You should see the following image!



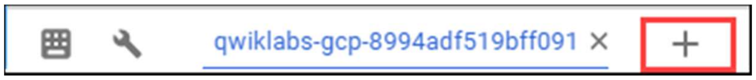
A list of my Cats

Mr. Whiskers is 4 years old!
Ms. Paws is 2 years old!

Open a separate Cloud Shell tab

While the Rails server is running, open a new Cloud Shell instance to run the remaining steps for this lab.

Open another Cloud Shell instance by using the **+** button to create a new tab:



Next, ssh into your `ruby-vm`

```
gcloud compute ssh --zone us-central1-a ruby-vm
```

Deployment Configuration

You now have a working Cloud SQL for PostgreSQL with Rails app. Let's deploy it to the App Engine flexible environment!

App Engine Flexible environment uses an `app.yaml` file to describe an application's deployment configuration. If this file is not present, the gcloud tool will try to guess the deployment configuration. However, it is a good idea to provide this file because Rails requires a `secret key` in production and App Engine requires the `connectionName` for the Cloud SQL instance.

Move into the directory created by Rails for the application:

```
cd /home/ruby_workspace/app_name
```

Create a new file called `app.yaml` and add the following to the file, replacing `[SECRET KEY]` with the generated secret key and `[YOUR_INSTANCE_CONNECTION_NAME]` with the generated connectionName of the Cloud SQL for PostgreSQL instance:

```
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]
```

When the app is deployed, the environment variable `SECRET_KEY_BASE` in production will be set with the `secret key` and App Engine will set up the Cloud SQL Proxy to communicate with the Cloud SQL for PostgreSQL instance.

Deploying the Application on App Engine

First, create an App Engine instance by using:

```
gcloud app create
```

Note: A prompt displays with a list of regions to select from. You can read more about Regions and Zones
After this command completes, deploy your app on App Engine:

```
gcloud app deploy
```

Note: First time deployment may take five to ten minutes. This is because App Engine Flexible environment will automatically provision a Compute Engine virtual machine for you behind the scenes, and then install the application, and start it.

Test Completed Task

Click **Check my progress** to verify your performed task. If you have completed the task successfully you will granted with an assessment score.

Deploying the Application on App Engine

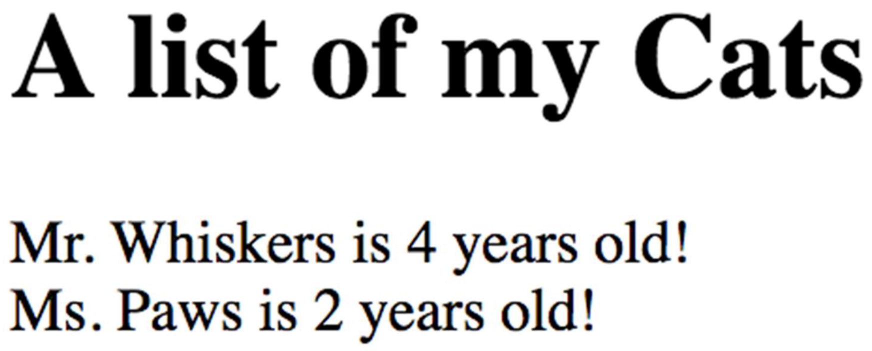
Check my progress

Check out your Rails app

After the application deploys, you can visit it by opening the URL

`https://[PROJECT_ID].appspot.com` in your web browser.

Note: You can also use the command `gcloud app browse` in Cloud Shell to provide you with the correct path to view your project.
You should see your List of Cats! Here's an example of what you should see:



A list of my Cats

Mr. Whiskers is 4 years old!
Ms. Paws is 2 years old!

Test your Understanding

Below are multiple-choice questions to reinforce your understanding of this lab's concepts. Answer them to the best of your abilities.

Ruby on Rails, or Rails, is a _____.
close~~Programming language~~
close~~Application~~
checkserver-side web application framework
Submit

Congratulations!

You learned how to use Cloud SQL for PostgreSQL using Ruby on Rails and then deploy it to the App Engine flexible environment.

Finish Your Quest

A Quest is a series of related labs that form a learning path. This self-paced lab is part of the Qwiklabs Quests Website and Web Application and Cloud SQL. Completing this Quest earns you the badge above to recognize your achievement. You can make your badge public and link to them in your online resume or social media account.
Enroll in Website and Web Application or Cloud SQL and get immediate completion credit if you've taken this lab. See other available Qwiklabs Quests.

Next Steps / Learn More

Continue your learning with another lab:

- Deploy Refinery CMS to App Engine Flexible Environment
- Ruby on Google Cloud
- Google Cloud SQL for PostgreSQL
- Google Cloud Ruby Gem

Google Cloud Training & Certification

...helps you make the most of Google Cloud technologies. Our classes include technical skills and best practices to help you get up to speed quickly and continue your learning journey. We offer fundamental to advanced level training, with on-demand, live, and virtual options to suit your busy schedule. Certifications help you validate and prove your skill and expertise in Google Cloud technologies.
Manual Last Updated September 10, 2020
Lab Last Tested September 10, 2020