Creating a PostgreSQL Database with Render

Learn how to create a PostgreSQL database using Render!

Introduction

One benefit of using Render as a **Platform as a Service** (**PaaS**) provider is the ability to create a PostgreSQL database that can be accessed and used within our deployed applications. Render even provides all of the connection information needed to access the newly created database, which in turn will allow us to create tables within the database that can be connected to, modified, and queried.

In this tutorial, we will walk through how to create a database on Render. This database will contain a single table that will hold text data. We will learn:

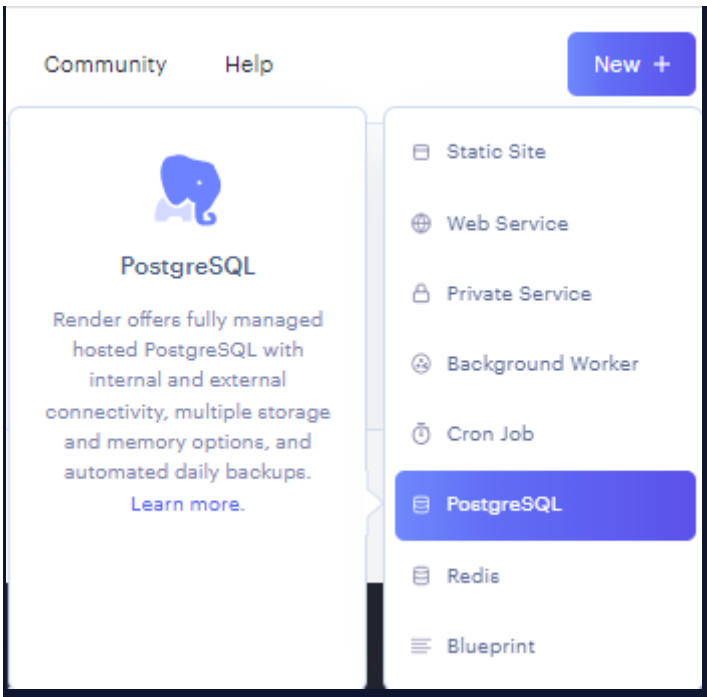
- Creating a PostgreSQL Database in Render: configure a new database instance using Render
- Connecting to the Database: connect to the newly created database using the Render-provided connection information
- Creating a Table: create a table within the database

Let's get started!

Creating a PostgreSQL Database in Render

Log into Render and navigate to the dashboard. From the top-right of the menu, click the blue, "New +" button to reveal a dropdown menu and then select "PostgreSQL" to set up our new database.

Please note, you cannot have more than one free tier active database at a time. If you find that you need multiple active databases, consider Render's paid offerings.



Let's go through the main settings to be aware of:

- **Name**: This field represents the unique name we want for our PostgreSQL instance. The name should be unique from any other PostgreSQL instances we have created under our Render account. For this tutorial, let's imagine we're creating a database instance to store text about activities. We'll set this field to `my-activity-database`.

- **Database**: This represents the name of the database. We'll set this field to `activity_database`.

- **User **: If we have a specific username we would like to create to access the database instance and tables, we can specify it here. Leave this field blank to generate a random username. We will use the username `activities_user` here.

- **Region**: This indicates the region where the PostgreSQL database service will run. In order to privately access our database, the region where we deployed our web service must match the region chosen here.

By having the resources in the same region, we can simply use the internal database URL to access the database. If we use a different region for the database, we would need to use the Render-provided external database URL to access the database, which can lead to decreased performance. We will discuss these URLs in a bit more detail soon.

- **PostgreSQL Version**: We can select the version of PostgreSQL that we want to use for our database. For this tutorial, we will use the current latest version. Note that for future deployed applications, they may require a different/specific version of PostgreSQL depending on the project needs.

- **Datadog API Key**: Since this tutorial will not cover Datadog monitoring, we can leave this field blank.

- **Instance Type**: Finally, there is a setting to select which instance type we will use for the database. Render offers a “Free” instance tier which provides sufficient resources for deploying our full-stack application. However, note that Render will expire free tier databases after 90 days and will not perform any automatic backups of the database. Explore the features and limitations of the free instance type in the Render documentation.

Free response

What are the benefits to keeping the Render PostgreSQL database in the same region as other Render-deployed web services?

Your response

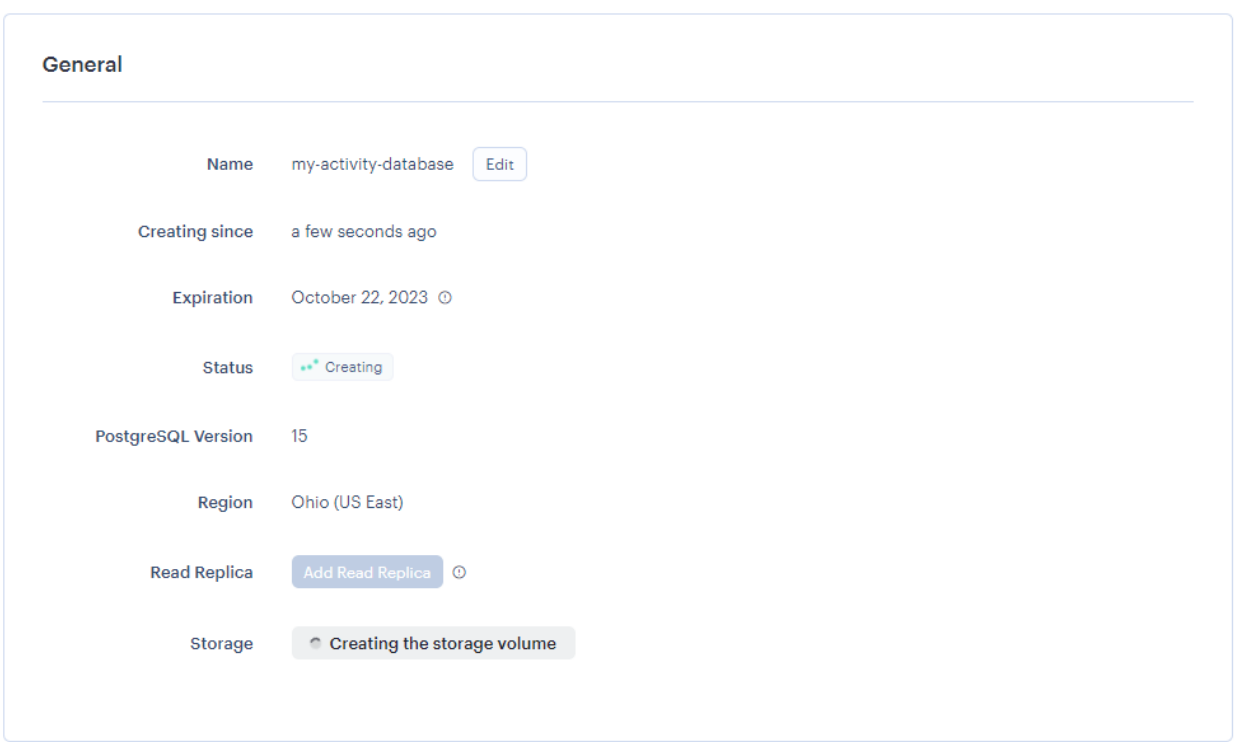
Performance. The data that the application needs for function correctly would be accessed faster if the database and the render-deployed web service are located in the same region.

Our answer

Ensuring that the PostgreSQL database and any deployed web services exist in the same region allows our web services to be able to seamlessly communicate with the database via the provided internal database URL. Having these instances in differing regions requires use of the external database URL to communicate with the database, which can lead to a decrease in application performance.

Now that our settings are configured, let's create our database! Click the blue "Create Database" button at the bottom of the page.

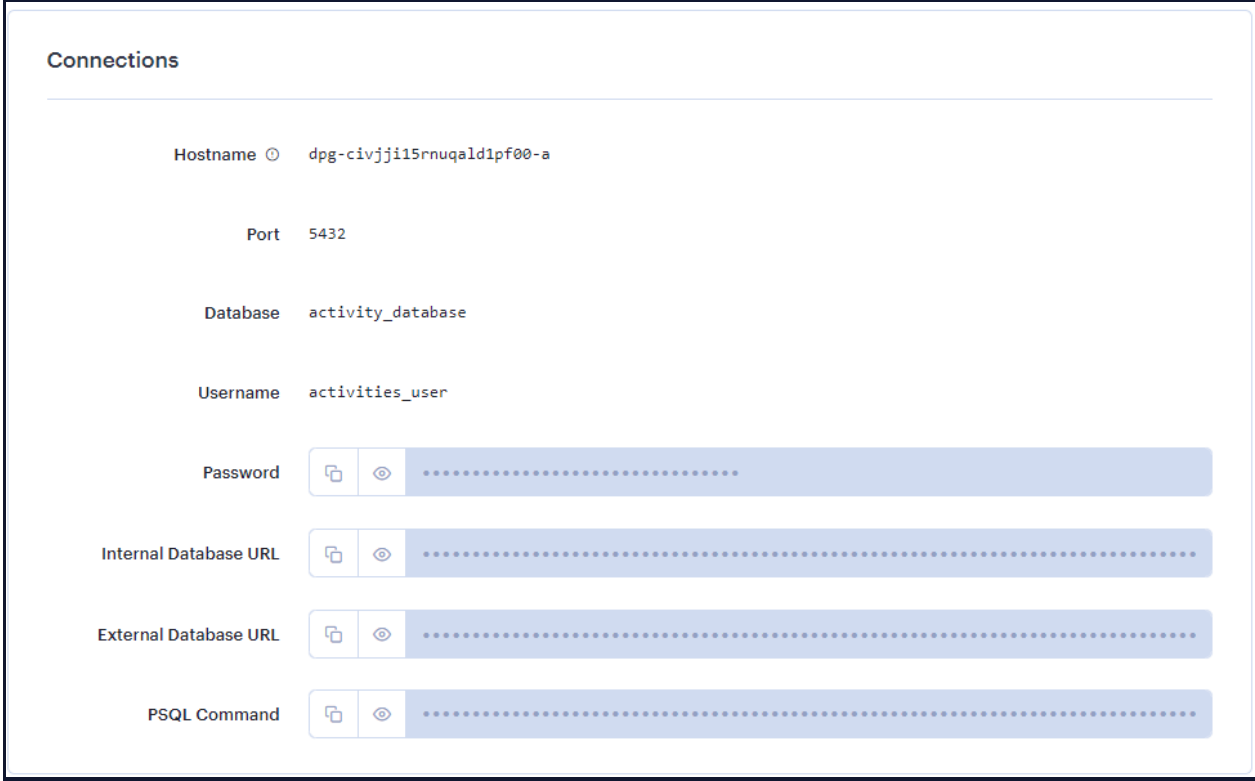
We'll see that the database is now in a "Creating" status. We can also easily view the 90-day expiration date in which our database will expire, as well as the settings we just configured earlier.



If we scroll down further, we will see a section called **Connections** that will detail how we can connect to our database. Once our database is ready, we can see that our database has a hostname and port number. We can also see the username we set earlier

and that our username now has a generated password that can be viewed. These credentials can be used to log in to the database locally via a terminal.

There are also two fields that provide URLs (that are starred out by default). One is an internal database URL, which can only be used if the deployed application and database are located in the same region. The internal database URL is a full connection string that provides the username, password, and table information all in one string. Make note of this internal URL as we will need it later to access the database from our source code. The external database URL is a full connection string that is used when we need to access our database from sources outside of Render (or from deployed applications that are not in the same region as our database). Conveniently, Render also provides a PSQL command that can be executed on the local computer's terminal in order to connect to the database instance. Since these provided connection URLs do contain sensitive information like our username and password, we should be sure to keep our connection information protected.



Now that we created a database, we need to add at least one database table so that we can add data to the table from our code.

Internal Database URL

postgres://activities_user:Fm4H9hJBtUoLOUbOt3C4wTRjRRTaHAm4@dpg-cnt7g9f109ks73b4ml1g-a/activity_database_afd3

PSQL Command

PGPASSWORD=Fm4H9hJBtUoLOUbOt3C4wTRjRRTaHAm4 psql -h dpg-cnt7g9f109ks73b4ml1g-a.oregon-postgres.render.com -U activities_user activity_database_afd3

```
PGPASSWORD=<password_goes_here> psql -h <hostname>.ohio-postgres.render.com -U
activities_user activity_database
```

activity_database

```
activity_database=>
```

my_activities

```
CREATE TABLE my_activities (activity text);
```

```
activity_database=> CREATE TABLE my_activities (activity text);
```

my_activities

activity text

CREATE TABLE

CREATE TABLE ;

\dt

```
\dt
```

my_activities

```
                List of relations
 Schema |     Name      | Type  |     Owner
--------+---------------+-------+----------------
 public | my_activities | table | ukvkznxdnpozbi
(1 row)
```

```
activity_database=> \q
```

After running this command, we will see that we have exited the PSQL console and have returned to our main terminal console.

Congratulations! You've created and set up your first PostgreSQL database through Render! Now you can access your database through your code to add, remove, and query data from your tables.