Community Tutorials

COMMUNITY HOME (/COMMUNITY) SEARCH TUTORIALS (/DOCS/TUTORIALS)

EDIT ON GITH
(HTTPS://GITHUB.COM/GOOGLECLOUDPLATFORM/COMMUNITY/EDIT/MASTER/TUTORIALS/SETTI
UP-POSTGRES-HOT-STANDBY.

REPORT ISSUE
(HTTPS://GITHUB.COM/GOOGLECLOUDPLATFORM/COMMUNITY/ISSUES/NEW?
TITLE=ISSUE%20WITH%20TUTORIALS/SETTING-UP-POSTGRES-HOT-
STANDBY.MD&BODY=ISSUE%20DESCRIPTION)

PAGE
(HTTPS://GITHUB.COM/GOOGLECLOUDPLATFORM/COMMUNITY/COMMITS/MASTER/TUTORIALS/
UP-POSTGRES-HOT-STAN

How to set up PostgreSQL for high availability and replication with Hot Standby

Author(s): @jimtravis (https://github.com/jimtravis), Published: 2017-02-18

★ Google Cloud Community tutorials submitted from the community do not represent official Google Cloud product documentation.

Learn how to configure PostgreSQL to run in Hot Standby mode on Compute Engine. You'll use two Compute Engine instances. One instance will run the primary PostgreSQL server and the other instance will run the standby server.

Alternatively, you can use Postgres as a service through Google Cloud SQL (/sql/docs/postgres).

For most applications, data is a critical commodity. Storing data in one place is a risky proposition, so you need to have a strategy and a plan in place (/solutions/designing-a-disaster-recovery-plan) to ensure that you can recover from a failure as quickly as possible. One way to help prevent data loss is to store the same data on multiple servers and keep those databases synchronized.

PostgreSQL, or Postgres, offers various ways (http://www.postgresql.org/docs/9.3/static/different-replication-solutions.html) to archive and replicate the primary database for backup, high- availability, and load balancing scenarios. In Hot Standby mode (https://www.postgresql.org/docs/9.3/static/hot-standby.html), the system employs two or more servers:

- A *primary* server runs the active database. This database accepts connections from clients and permits read-write operations.
- One or more *standby* servers run a copy of the active database. These databases are configured to accept connections from clients and permit read-only operations. If the primary database server fails, the system can *fail over* to the standby server, which makes the standby server the active primary server.

The rest of this tutorial will use and discuss a single standby server.

Understanding Hot Standby

Here are some basic points you should understand about how Hot Standby works:

- Postgres uses write-ahead logging (WAL) to continuously archive (https://www.postgresql.org/docs/9.3/static/continuous-archiving.html) database transactions. For each change made to the data files, WAL writes an entry in a log file. The system uses these log entries to perform point-in- time restoration from archives and to keep the standby server up to date. This means that when you set up Hot Standby, you're also setting up archiving.
- The process of updating the standby server with WAL entries is called streaming replication (http://www.postgresql.org/docs/9.3/static/warm-standby.html#STREAMING-REPLICATION)

. This process operates asynchronously, which means it can take some time for the standby server to receive an update from the primary server. Though this delay can be very short, synchronization between the servers is not instantaneous. If your application requires strict consistency between the database instances, you should consider another approach.

- Postgres doesn't provide functionality to automatically fail over when the primary server fails. This is a manual operation unless you use a third-party solution to manage failover.
- Load balancing is not automatic with Hot Standby. If load balancing is a requirement for your application, you must provide a load-balancing solution that uses the primary server for read-write operations and the standby server for read-only operations.

Important: This tutorial covers a basic setup of two servers in a Hot Standby configuration. The tutorial doesn't try to cover every configuration that is available to you for this scenario. For a complete understanding of how to optimize Postgres in standby scenarios, refer to the Postgres documentation (https://www.postgresql.org/docs/9.1/static/high-availability.html).

Objectives

- Set up two Compute Engine instances running Postgres.
- Create a new table for a guestbook app.
- Configure the primary server.
- Back up the primary server to the standby server.
- Configure the standby server to run in Hot Standby mode.
- Start the standby server and test it.

Before you begin

Select or create a Google Cloud Platform project (https://console.cloud.google.com/project).

Costs

This tutorial uses billable components of Google Cloud Platform (GCP), including Compute Engine.

Use the Pricing Calculator (/products/calculator) to generate a cost estimate based on your projected usage.

Setting up the Compute Engine instances

To create the primary and standby servers, follow the steps in How to Set Up PostgreSQL on Compute Engine (/community/tutorials/setting-up-postgres). Note the following items:

- You won't need to use pgAdmin for this tutorial.
- Follow the installation steps twice, once for each server.
- Follow the steps in **Connecting remotely** (/community/tutorials/setting-up-postgres#connecting-remotely).
- You'll need to open the network port only once, not once for each server.
- Note the IP address of each server. You'll need these values when you modify the configuration files.
- Keep an SSH window open for each server as you work through this tutorial.

Creating the guestbook table

On the primary server, create a simple table for testing purposes. This table stores entries for a website's guestbook, where visitors to the site can leave a comment. The data fields include the vistor's email address, a serial ID, the current date and time, and a text message written by the visitor.

In the SSH terminal for the primary server:

1. Run the root shell:

```
$ sudo -s
```

2. Run PSQL as user `postgres` and access the database named `postgres`:

```
$ sudo -u postgres psql postgres
```

3. At the PSQL prompt, enter the following command to create the table:

```
CREATE TABLE guestbook (visitor_email text, vistor_id serial, date ti
```

4. Add an entry to the table:

```
INSERT INTO guestbook (visitor_email, date, message) VALUES ( 'jim@gm
```

   You should see a confirmation message that says: `INSERT 0 1`.

5. Enter `\q` to exit PSQL.

Don't exit the root shell. You'll use the root shell throughout this tutorial.

Configuring the primary server

To configure the primary server, you will:

- Create a Postgres user for replication activities.
- Create a directory to store archive files.
- Edit two configuration files: `pg_hba.conf` and `postgresql.conf`.

Create a user for replication

To perform replication, Postgres requires a user, also called a *role*, with special permissions. On the primary server, run the following command:

```
$ sudo -u postgres createuser -U postgres repuser -P -c 5 --replication
```

This command performs the following actions:

- `sudo -u postgres` ensures that the `createuser` command runs as the user `postgres`. Otherwise, Postgres will try to run the command by using peer authentication, which means the command will run under your Ubuntu user account. This account probably doesn't have the right privileges to create the new user, which would cause an error.
- The `-U` option tells the `createuser` command to use the user `postgres` to create the new user.
- The name of the new user is `repuser`. You'll enter that username in the configuration files.
- `-P` prompts you for the new user's password. **Important**: For any system with an Internet connection, use a strong password to help keep the system secure.
- `-c` sets a limit for the number of connections for the new user. The value `5` is sufficient for replication purposes.
- `--replication` grants the `REPLICATION` privilege to the user named `repuser`.

Create the archive directory

Create a directory to store archive files. This directory is a subdirectory of the cluster's data directory, which is named `main` by default. You'll use this path in one of the configuration files. If you have configured your cluster to use a different directory for data, you must create your archive directory in that directory and then change the corresponding configuration setting.

In the SSH terminal for the primary server, enter the following command:

```
$ mkdir -p ../../var/lib/postgresql/main/mnt/server/archivedir
```

Edit `pg_hba.conf`

This configuration file contains the settings for client authentication. You must add an entry for the user `repuser` to enable replication.

1. Edit the file. For PostgreSQL version 9.3, you can enter::

```
$ nano ../../etc/postgresql/9.3/main/pg_hba.conf
```

2. After the example replication entries, add the following lines. Replace `[standby-IP]` with the external IP address of the standby server:

```
# Allow replication connections
host     replication     repuser         [standby-IP]/32        md5
```

3. Save and close the file.

Edit `postgresql.conf`

This configuration file contains the main settings for Postgres. Here, you will modify the file to enable archiving and replication.

★ **Important:** Don't forget to uncomment any lines you edit in the configuration files, or your changes won't take effect.

1. Edit the file. In the terminal for the primary server, enter the following command:

```
$ nano ../../etc/postgresql/9.3/main/postgresql.conf
```

2. In the **WRITE AHEAD LOG** section, in the **Settings** section, change the WAL level:

```
wal_level = hot_standby
```

3. In the **Archiving** section, change the archive mode:

```
archive_mode = on
```

4. Change the value for the archive command. This setting tells Postgres to write the archive files to the directory that you created in a previous step:

```
archive_command = 'test ! -f mnt/server/archivedir/%f && cp %p mnt/se
```

5. In the **REPLICATION** section, in the **Sending Server(s)** section, change the value for the maximum number of WAL sender processes:

```
max_wal_senders = 3
```

   For this tutorial, the value of `3` is sufficient to enable backup and replication.

6. Save and close the file.

Restart the primary server

Now that you've made the configuration changes, restart the server to make the changes effective. Enter the following command:

```
$ sudo service postgresql restart
```

Backing up the primary server to the standby server

Before making changes on the standby server, stop the service. In the SSH terminal for the standby server, run the following command:

```
$ sudo service postgresql stop
```

★ **Important:** Don't start the service again until all configuration and backup steps are complete. You must bring up the standby server in a state where it is ready to be a backup server. This means that all configuration settings must be in place and the databases must be already synchronized. Otherwise, streaming replication will fail to start.

Run the backup utility

The backup utility, named `pg_basebackup`, will copy files from the data directory on the primary server to the same directory on the standby server.

1. Make sure you're running commands in the root shell. In the SSH terminal for the standby server, enter the following command:

   ```
   $ sudo -s
   ```

   Continue to use the root shell for the remainder of this tutorial.

2. The backup utility won't overwrite existing files, so you must rename the data directory on the standby server. Run the following command:

   ```
   $ mv ../../var/lib/postgresql/9.3/main ../../var/lib/postgresql/9.3/m
   ```

3. Run the backup utility. Replace `[primary-IP]` with the external IP address of the primary server.

   ```
   $ sudo -u postgres pg_basebackup -h [primary IP] -D /var/lib/postgres
   ```

   The backup utility will prompt you for the password for the user named `repuser`.

The backup process should take just a few moments. When it's done, you can move on to configuring the standby server.

Configuring the standby server

To configure the standby server, you'll edit `postgresql.conf` and create a new configuration file named `recovery.conf`.

Edit `postgresql.conf`

For the standby server, you only need to change one setting in this file. Follow these steps:

1. Edit the file. In the terminal for the standby server, enter the following command:

```
$ nano ../../etc/postgresql/9.3/main/postgresql.conf
```

2. In the **REPLICATION** section, in the **Standby Servers** section, turn on Hot Standby and uncomment the line:

```
hot_standby = on
```

3. Save and close the file.

Create the recovery configuration file

When you implement a server in Hot Standby mode, you must supply a configuration file that contains the settings that will be used in the event of data recovery. This file is named `recovery.conf`. To add this file to the standby server, follow these steps:

1. Copy the sample recovery file to the proper location. In the terminal for the standby server, enter the following command:

```
$ cp -avr ../../usr/share/postgresql/9.3/recovery.conf.sample /../../
```

2. Edit the recovery file:

```
$ nano /../../var/lib/postgresql/9.3/main/recovery.conf
```

3. In the **STANDBY SERVER PARAMETERS** section, change the standby mode:

```
standby_mode = on
```

4. Set the connection string to the primary server. Replace `[primary-external-IP]` with the external IP address of the primary server. Replace `[password]` with the password for the user named `repuser`.

```
primary_conninfo = 'host=[primary-external-IP] port=5432 user=repuser
```

5. (Optional) Set the trigger file location:

```
trigger_file = '/tmp/postgresql.trigger.5432'
```

   The `trigger_file` path that you specify is the location where you can add a file when you want the system to fail over to the standby server. The presence of the file "triggers" the failover. Alternatively, you can use the `pg_ctl promote` command to trigger failover.

6. Save and close the file.

Start the standby server

You now have everything in place and are ready to bring up the standby server. In the terminal for the standby server, enter the following command:

```
$ service postgresql start
```

Seeing the replication at work

To demonstrate that the replication between the primary and standby servers is working, you can add a row to the `guestbook` table on the primary server and then query the standby server to see the new row.

Recall that you have already added one row to the table on the primary server. Start by verifying that the standby server has the same information.

1. On the standby server, start PSQL:

   ```
   $ sudo -u postgres psql postgres
   ```

2. At the PSQL prompt, enter the following query:

   ```
   select * from guestbook;
   ```

   You should see that the table contains the single row that you originally added. Now, add a second row on the primary server.

3. On the primary server, start PSQL:

   ```
   $ sudo -u postgres psql postgres
   ```

4. At the PSQL prompt, enter the following command:

   ```
   INSERT INTO guestbook (visitor_email, date, message) VALUES ('jim@gma
   ```

5. Switch back to the standby server terminal and repeat the query for all rows of the guestbook:

   ```
   select * from guestbook;
   ```

You should now see that the standby server has received the update from the primary server.

6. To exit PSQL, enter `\q`.
7. To exit the root shells, enter `exit` in each terminal window.

Troubleshooting

After completing all the steps, if you're not seeing the data replicate, you might have missed a step or some small detail. Common mistakes include:

- Leaving a setting commented out.
- Forgetting to replace placeholder text in a setting or command. For example, some settings require a host IP address or a password.
- Entering the wrong IP address or password.
- Using a primary server setting for the standby server or vice-versa.

If you find yourself in this state, here are the steps to follow:

1. Look at the Postgres log on each server. These logs can contain information that will help you troubleshoot the issue.

```
$ less ../../var/log/postgresql/postgresql-9.3-main.log
```

2. Check the primary server settings. If there are mistakes, fix them and then restart the server.
3. Shut down the standby server.
4. Check the standby server settings and correct them if needed.
5. On the standby server, rename the `main` folder to something new, such as `main_old_2`:

```
$ mv ../../var/lib/postgresql/9.3/main ../../var/lib/postgresql/9.3/m
```

6. On the standby server, run `pgbasebackup` again to synchronize the data. Substitute `[primary-IP]` with your primary server's external IP address:

```
$ sudo -u postgres pg_basebackup -h [primary-IP] -D /var/lib/postgres
```

7. The `main` folder now needs a copy of `recovery.conf`. You can simply copy it from the folder that you renamed to `main_old_2`:

```
$ cp ../../var/lib/postgresql/9.3/main_old_2/recovery.conf ../../var/
```

8. Start the standby server.
9. Retest the replication functionality.

Cleaning up

After you've finished the tutorial, you can clean up the resources you created on Google Cloud Platform so you won't be billed for them in the future. The following sections describe how to delete or turn off these resources.

Deleting the project

The easiest way to eliminate billing is to delete the project you created for the tutorial. If you don't want to delete the project, delete the individual instances, as described in the next section.

Warning: Deleting a project has the following consequences:

- If you used an existing project, you'll also delete any other work you've done in the project.
- You can't reuse the project ID of a deleted project. If you created a custom project ID that you plan to use in the future, you should delete the resources inside the project instead. This ensures that URLs that use the project ID, such as an appspot.com URL, remain available.

If you are exploring multiple tutorials and quickstarts, reusing projects instead of deleting them prevents you from exceeding project quota limits.

To delete the project:

1. In the Cloud Platform Console, go to the Projects page (https://console.cloud.google.com/iam-admin/projects).
2. In the project list, select the project you want to delete and click **Delete project**. After selecting the checkbox next to the project name, click **Delete project**.
3. In the dialog, type the project ID, and then click **Shut down** to delete the project.

Deleting instances

To delete a Compute Engine instance:

1. In the Cloud Platform Console, go to the **VM Instances** page (https://console.cloud.google.com/compute/instances).
2. Click the checkbox next to your `lamp-tutorial` instance.
3. Click the **Delete** button at the top of the page to delete the instance.

Deleting firewall rules

Deleting firewall rules for the default network

To delete a firewall rule:

1. In the Cloud Platform Console, go to the **Firewall Rules** page (https://console.cloud.google.com/networking/firewalls/list).
2. Click the checkbox next to the firewall rule you want to delete.
3. Click the **Delete** button at the top of the page to delete the firewall rule.

Next steps

- Explore the PostgreSQL documentation (https://www.postgresql.org/docs/).

Submit a Tutorial

Share step-by-step guides

SUBMIT A TUTORIAL (/COMMUNITY/TUTORIALS/WRITE)

Request a Tutorial

Ask for community help

SUBMIT A REQUEST (HTTPS://GITHUB.COM/GOOGLECLOUDPLATFORM/COMMUNITY/ISSUES?Q=IS%3AOPEN+IS%3AISSUE+LABEL%3A%22TUTORIAL+REQUEST%22)

GCP Tutorials

Tutorials published by GCP

VIEW TUTORIALS (/DOCS/TUTORIALS)