How to Move PostgreSQL Data Directory to Cloud Volumes Service for AWS

Prabu Arjunan, NetApp
July 2020

TABLE OF CONTENTS

1 Introduction

PostgreSQL (also called Postgres) is a free, open-source, relational database management system (RDBMS) that emphasizes extensibility and SQL compliance. PostgreSQL has earned a strong reputation for stability, power, reliability, feature robustness, and performance. It is backed by more than 30 years of community development, which has contributed to its high levels of resilience, integrity, and reliability.

Amazon Web Services (AWS) is an excellent platform for running traditional RDBMSs. Using NetApp® Cloud Volumes Service for AWS as the storage for PostgreSQL provides enhanced availability, instant copies to accelerate development, and lower costs. The design of your PostgreSQL installation on Amazon Elastic Compute Cloud (EC2) depends largely on the scale at which you are operating.

1.1 Prerequisites

Before continuing with this tutorial, make sure that you have already subscribed to NetApp Cloud Volumes Service for AWS and have created an NFS volume.

Manage Cloud Volumes

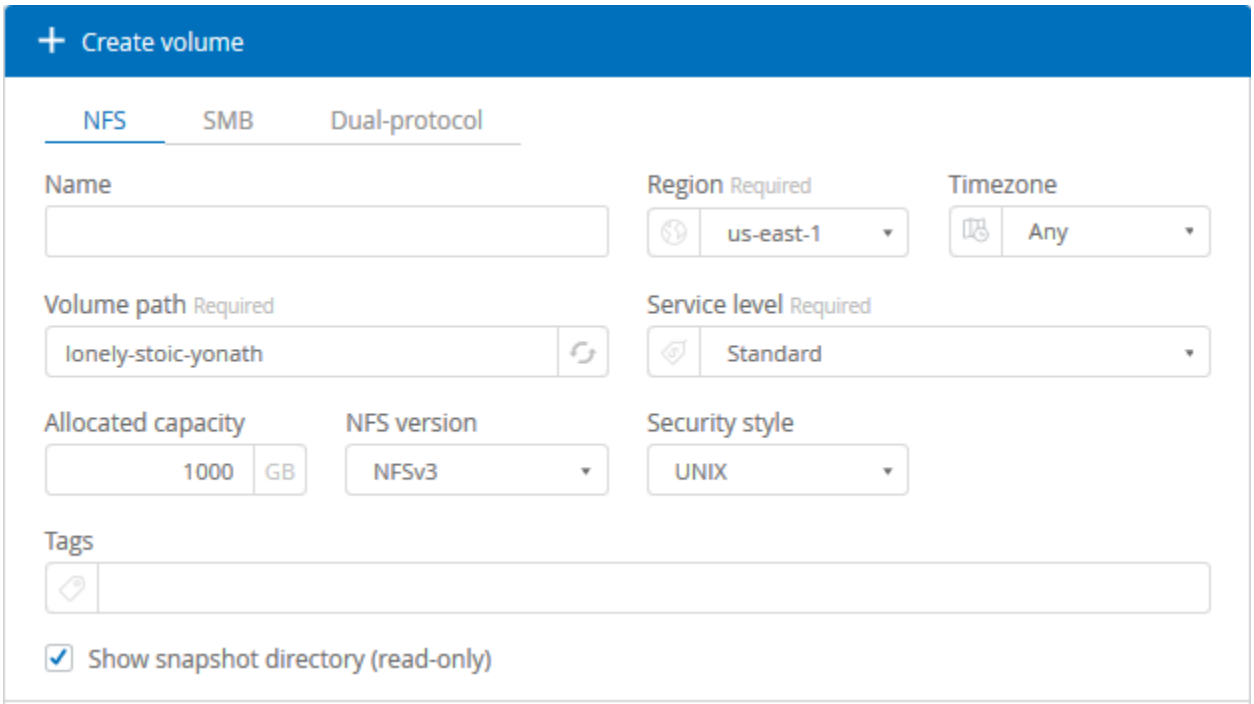
To manage your volume or to create additional cloud volumes, follow the instructions at NetApp Cloud Volumes Service for AWS Documentation. For example, you can create and mount a cloud volume. Also, make sure that you have installed PostgreSQL on Ubuntu 18.04. There are multiple sources for installing PostgreSQL on Ubuntu. I used this open source link to install the following version of psql.

```
postgres@ip-10-1-0-249:~$ psql
psql (12.2 (Ubuntu 12.2-2.pgdg18.04+1))
```

1.2 Provision an NFS Volume for PostgreSQL DB

Complete the fields at the top of the Create Volume page to define the volume name, size, service level, and so on.

1. Click Create New Volume after you have logged into the NetApp Cloud Orchestrator site with the email address that you provided during your subscription and you have selected the region.

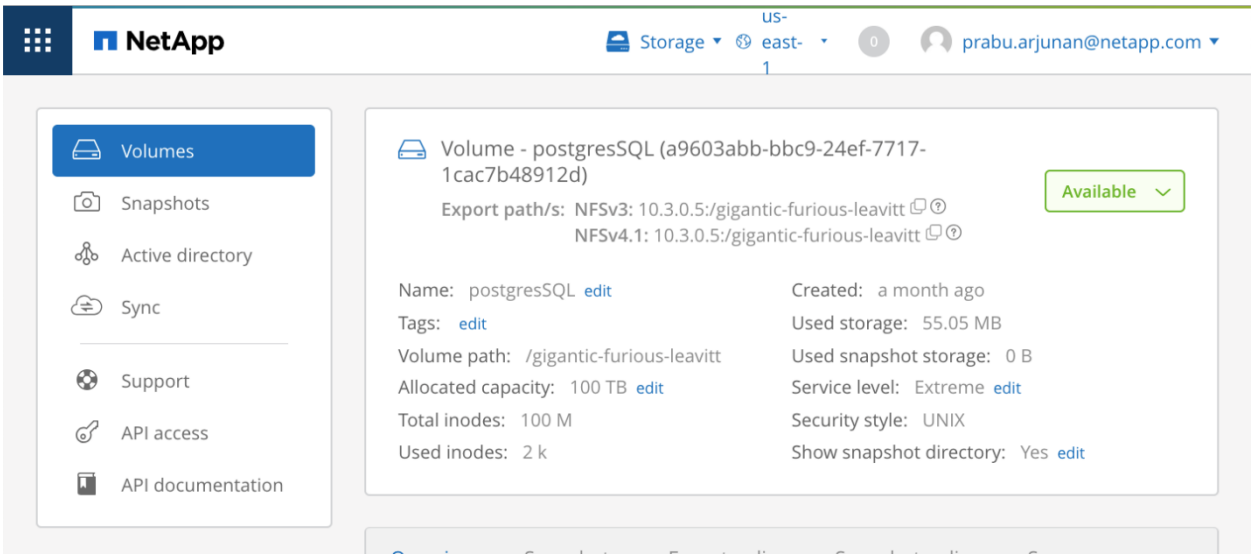


2. On the Create Volume page, select NFS, SMB, or Dual-Protocol as the protocol for the volume you are creating.
3. In the Name field, specify the volume name.
4. In the Region field, select the AWS region in which to create the volume. This region must match the region you configured on AWS.
5. In the Timezone field, select a time zone.
6. In the Volume Path field, specify the path to use or accept the automatically generated path.
7. In the Service Level field, select the level of performance for the volume: Standard, Premium, or Extreme. See Selecting the Service Level for details.
8. In the Allocated Capacity field, select the capacity required. Note that the number of available inodes depends on allocated capacity. See Selecting the Allocated Capacity for details.
9. In the NFS Version Field, select NFSv3, NFSv4.1, or Both, depending on your requirements.
10. If you selected Dual-Protocol, you can select the security style by selecting NTFS or UNIX from the drop-down menu in the Security Style field.
11. Security styles affect the file permission type used and how permissions can be modified.
    a. UNIX uses NFSv3 mode bits, and only NFS clients can modify permissions.
    b. NTFS uses NTFS ACLs, and only SMB clients can modify permissions.
12. In the Show Snapshot Directory field, keep the default so that you can view the Snapshot directory for this volume, or uncheck the box to hide the list of Snapshot copies.

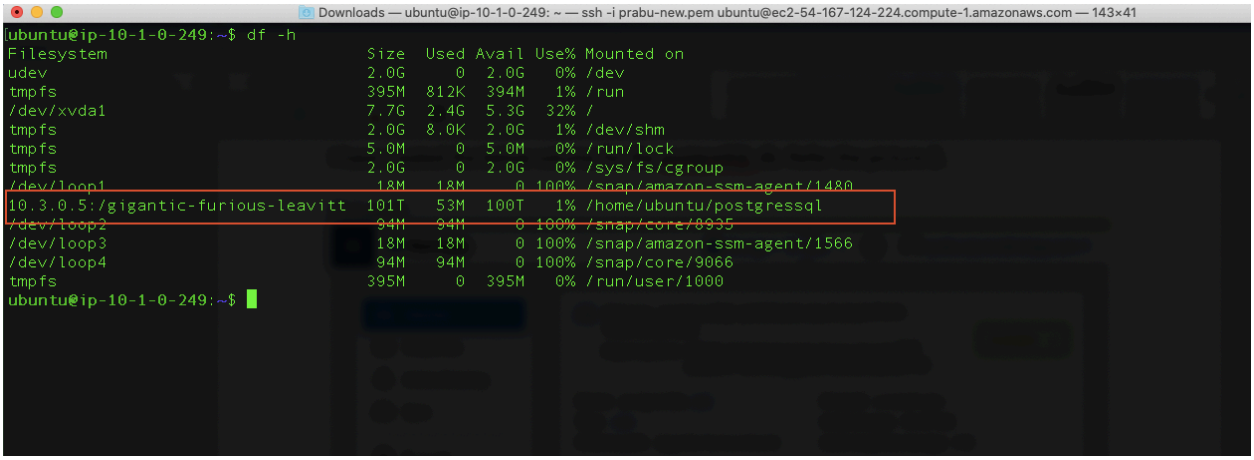
For full details, see Creating a Cloud Volume.

1.3 The Volume Created and Mounted

The following screenshot shows a volume for the PostgreSQL database.



As shown in the following screenshot, the volume is mounted in the Ubuntu client.



This completes PostgreSQL installation and Cloud Volumes Service volume provisioning and mounting. The next step is to move the PostgreSQL Data Directory into the newly created volume.

1.4 Moving the PostgreSQL Data Directory

First, verify the current location by starting an interactive PostgreSQL session. In the following command, `psql` is the command to enter the interactive monitor and `-u postgres` tells `sudo` to execute `psql` as the system's `postgres` user:

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

When you have the PostgreSQL prompt open, use the following command to show the current data directory:

```
$ postgres=# SHOW data_directory;
Output
               data_directory
-------------------------------------------
/var/lib/postgresql/12/main
```

To ensure the integrity of the data, stop PostgreSQL before you make changes to the data directory.

```
#sudo systemctl stop postgresql
```

`systemctl` does not display the outcome of all service management commands. To verify that you have successfully stopped the service, run the following command:

```
#sudo systemctl status postgresql
```

The final line of the output should tell you that PostgreSQL has been stopped:

```
Output
. . .
Apr 20 22:29:28 ip-10-1-0-249 systemd[1]: Stopped PostgreSQL RDBMS
```

Now that the PostgreSQL server is shut down, you can copy the existing database directory to the new location with `rsync`. Using the `-a` flag preserves the permissions and other directory properties, while `-v` provides verbose output so that you can follow the progress.

Start `rsync` from the `postgresql` directory to mimic the original directory structure in the new location. By creating that `postgresql` directory in the mount-point directory and retaining ownership by the PostgreSQL user, you can avoid permissions problems with future upgrades.

Note: Be sure that there is no trailing slash on the directory, which might be added if you use tab completion. If you do include a trailing slash, `rsync` dumps the contents of the directory into the mount point instead of copying over the directory itself.

The version directory, 10, isn't strictly necessary, because you've defined the location explicitly in the `postgresql.conf` file. However, following the project convention wouldn't hurt, especially if there's a need in the future to run multiple versions of PostgreSQL.

```
sudo rsync -av /var/lib/postgresql /home/ubuntu/postgressql
```

When the copy is complete, rename the current folder with a `.bak` extension and keep it until you have confirmed that the move was successful. This helps to avoid confusion that could arise from having similarly named directories in both the new and the old location.

```
sudo mv /var/lib/postgresql/12/main /var/lib/postgresql/12/main.bak
```

Now you are ready to configure PostgreSQL to access the data directory in its new location.

1.5 Pointing to the New Data Location

By default, the `data_directory` is set to `/var/lib/postgresql/12/main` in the `/etc/postgresql/12/main/postgresql.conf` file. Edit this file to reflect the new data directory.

```
sudo vi /etc/postgresql/12/main/postgresql.conf
```

Find the line that begins with `data_directory` and change the path that follows to reflect the new location. In the context of this tutorial, the updated directive looks like this:

```
/etc/postgresql/12/main/postgresql.conf
. . .
data_directory = '/home/ubuntu/postgressql/postgresql/12/main'. . .
```

Save and close the file by pressing `CtrlL + X`, `Y`, and then `E`nter. That is all you need to do to configure PostgreSQL to use the new data directory location. All that is left to do at this point is to start the PostgreSQL service again and check that it is pointing to the correct data directory.

1.6 Restarting PostgreSQL

After changing the `data-directory` directive in the `postgresql.conf` file, start the PostgreSQL server using `systemctl`.

```
sudo systemctl start postgresql
```

To confirm that the PostgreSQL server started successfully, check its status by again using `systemctl`:

```
sudo systemctl status postgresql
```

If the service started correctly, you can see the following line at the end of this command's output:

```
Output
Apr 20 22:29:28 ip-10-1-0-249 systemd[1]: Started PostgreSQL RDBMS
```

Finally, to make sure that the new data directory is in use, open the PostgreSQL command prompt.

```
sudo -u postgres psql
```

Check the value for the data directory again.

```
postgres=# SHOW data_directory;
Output
data_directory
-----------------------------------------
```

```
/home/ubuntu/postgressql/postgresql/12/main
```

This confirms that PostgreSQL is using the new data directory location. Now take a moment to make sure that you are able to access your database as well as interact with the data in it. When you have verified the integrity of your existing data, you can remove the backup data directory.

```
sudo rm -Rf /var/lib/postgresql/12/main.bak
```

1.7 Conclusion

After completing the steps described in this post, you will have successfully moved your PostgreSQL data directory to a new location residing on a Cloud Volumes Service volume.

About NetApp

NetApp is the leader in cloud data services, empowering global organizations to change their world with data. Together with our partners, we are the only ones who can help you build your unique data fabric. Simplify hybrid multicloud and securely deliver the right data, services, and applications to the right people at the right time. Learn more at www.netapp.com.