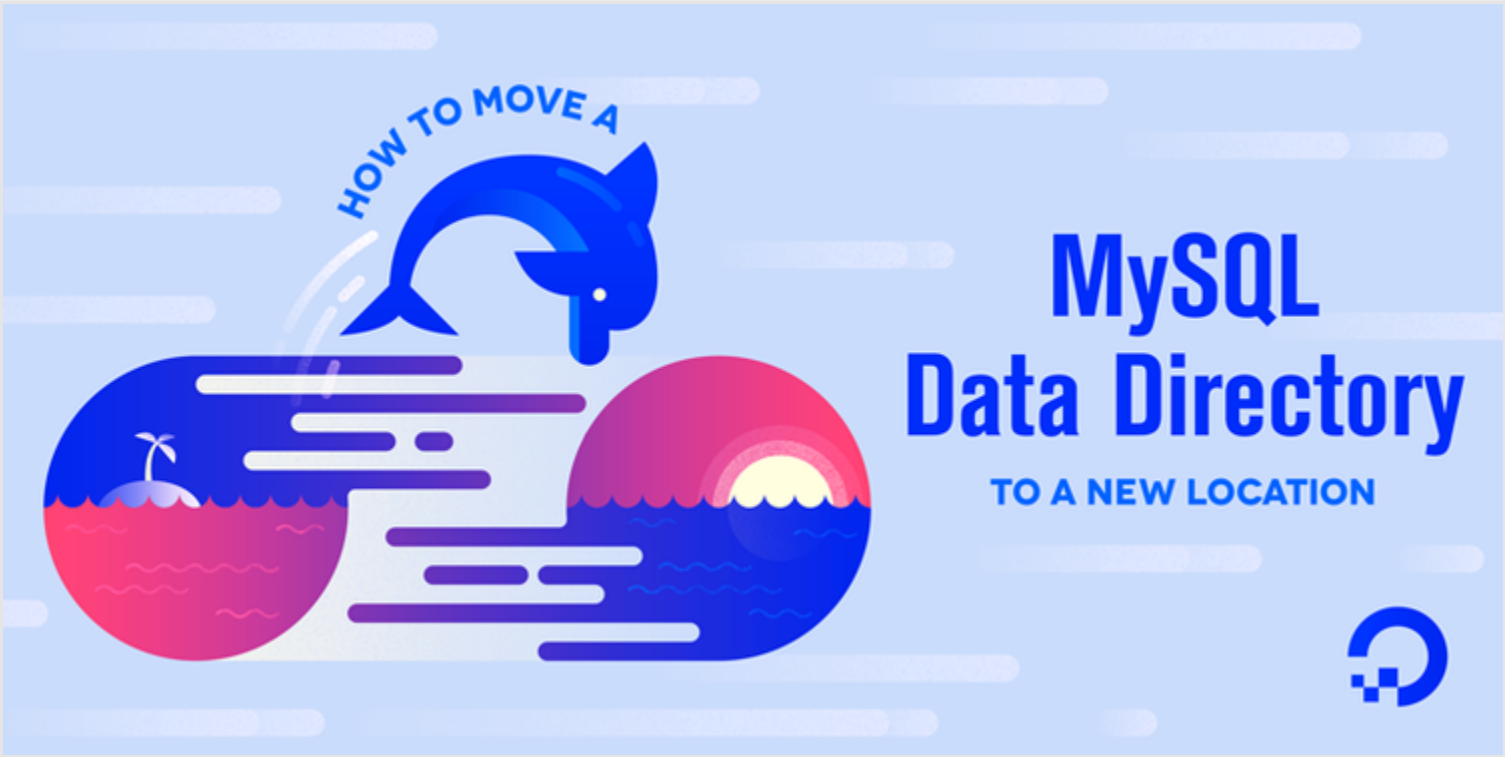

How To Move a MySQL Data Directory to a New Location on Ubuntu 18.04

Posted July 6, 2018 6.2k MYSQL DATABASES BLOCK STORAGE STORAGE UBUNTU UBUNTU 18.04

By: Melissa Anderson By: Hanif Jetha

Not using **Ubuntu 18.04**? Choose a different version:

Introduction

Databases grow over time, sometimes outgrowing the space on the file system. You can also run into I/O contention when they're located on the same partition as the rest of the operating system. RAID, network block storage, and other devices can offer redundancy and other desirable features. Whether you're adding more space, evaluating ways to optimize performance, or looking to take advantage of other storage features, this tutorial will guide you through relocating MySQL's data directory.

Prerequisites

To complete this guide, you will need:

- **An Ubuntu 18.04 server with a non-root user with `sudo` privileges**. You can learn more about how to set up a user with these privileges in our Initial Server Setup with Ubuntu 18.04 guide.
- **A MySQL server**. If you haven't already installed MySQL, the How To Install MySQL on Ubuntu 18.04 guide can help you.

In this example, we're moving the data to a block storage device mounted at `/mnt/volume-nyc1-01`. You can learn how to set one up in the How To Use Block Storage on DigitalOcean guide.

No matter what underlying storage you use, this guide can help you move the data directory to a new location.

Step 1 — Moving the MySQL Data Directory

To prepare for moving MySQL's data directory, let's verify the current location by starting an interactive MySQL session using the administrative credentials.

```
$ mysql -u root -p
```

When prompted, supply the MySQL root password. Then from the MySQL prompt, select the data directory:

```
mysql> select @@datadir;
```

```
Output
+-----------------+
| @@datadir       |
+-----------------+
| /var/lib/mysql/ |
+-----------------+
1 row in set (0.00 sec)
```

This output confirms that MySQL is configured to use the default data directory, `/var/lib/mysql/`, so that's the directory we need to move. Once you've confirmed this, type `exit` to leave the monitor.

To ensure the integrity of the data, we'll shut down MySQL before we actually make changes to the data directory:

```
$ sudo systemctl stop mysql
```

`systemctl` doesn't display the outcome of all service management commands, so if you want to be sure you've succeeded, use the following command:

```
$ sudo systemctl status mysql
```

You can be sure it's shut down if the final line of the output tells you the server is stopped:

```
Output
. . .
Jul 18 11:24:20 ubuntu-512mb-nyc1-01 systemd[1]: Stopped MySQL Community Server.
```

Now that the server is shut down, we'll 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 you can follow the progress.

> **Note:** Be sure there is no trailing slash on the directory, which may be added if you use tab completion. When there's a trailing slash, `rsync` will dump the contents of the directory into the mount point instead of transferring it into a containing `mysql` directory:

```
$ sudo rsync -av /var/lib/mysql /mnt/volume-nyc1-01
```

Once the `rsync` is complete, rename the current folder with a .bak extension and keep it until we've confirmed the move was successful. By re-naming it, we'll avoid confusion that could arise from files in both the new and the old location:

```
$ sudo mv /var/lib/mysql /var/lib/mysql.bak
```

Now we're ready to turn our attention to configuration.

Step 2 — Pointing to the New Data Location

MySQL has several ways to override configuration values. By default, the `datadir` is set to `/var/lib/mysql` in the `/etc/mysql/mysql.conf.d/mysqld.cnf` file. Edit this file to reflect the new data directory:

```
$ sudo nano /etc/mysql/mysql.conf.d/mysqld.cnf
```

Find the line that begins with `datadir=` and change the path which follows to reflect the new location.

In our case, the updated file looks like the output below:

/etc/mysql/mysql.conf.d/mysqld.cnf

```
. . .
datadir=/mnt/volume-nyc1-01/mysql
```

```
. . .
```

This seems like the right time to bring up MySQL again, but there's one more thing to configure before we can do that successfully.

Step 3 — Configuring AppArmor Access Control Rules

We'll need to tell AppArmor to let MySQL write to the new directory by creating an alias between the default directory and the new location. To do this, edit the AppArmor `alias` file:

```
$ sudo nano /etc/apparmor.d/tunables/alias
```

At the bottom of the file, add the following alias rule:

/etc/apparmor.d/tunables/alias

```
. . .
alias /var/lib/mysql/ -> /mnt/volume-nyc1-01/mysql/,
. . .
```

For the changes to take effect, restart AppArmor:

```
$ sudo systemctl restart apparmor
```

Note: If you skipped the AppArmor configuration step, you would run into the following error message:

```
Output
Job for mysql.service failed because the control process
exited with error code. See "systemctl status mysql.service"
and "journalctl -xe" for details.
```

The output from both `systemctl` and `journalctl` concludes with:

```
Output
Jul 18 11:03:24 ubuntu-512mb-nyc1-01 systemd[1]:
mysql.service: Main process exited, code=exited, status=1/FAILURE
```

Since the messages don't make an explicit connection between AppArmor and the data directory, this error can take some time to figure out.

Step 4 — Restarting MySQL

The next step is to start MySQL, but if you do, you'll run into another error. This time, instead of an AppArmor issue, the error happens because the script `mysql-systemd-start` checks for the existence of either a directory, `-d`, or a symbolic link, `-L`, that matches two default paths. It fails if they're not found:

/usr/share/mysql/mysql-systemd-start

```
. . .
if [ ! -d /var/lib/mysql ] && [ ! -L /var/lib/mysql ]; then
 echo "MySQL data dir not found at /var/lib/mysql. Please create one."
 exit 1
fi

if [ ! -d /var/lib/mysql/mysql ] && [ ! -L /var/lib/mysql/mysql ]; then
 echo "MySQL system database not found. Please run mysql_install_db tool."
 exit 1
fi

. . .
```

Since we need these to start the server, we will create the minimal directory structure to pass the script's environment check.

```
$ sudo mkdir /var/lib/mysql/mysql -p
```

Now we're ready to start MySQL.

```
$ sudo systemctl start mysql
$ sudo systemctl status mysql
```

To make sure that the new data directory is indeed in use, start the MySQL monitor.

```
$ mysql -u root -p
```

Look at the value for the data directory again:

Output

```
+----------------------------+
| @@datadir                  |
+----------------------------+
| /mnt/volume-nyc1-01/mysql/ |
+----------------------------+
1 row in set (0.01 sec)
```

Now that you've restarted MySQL and confirmed that it's using the new location, take the opportunity to ensure that your database is fully functional. Once you've verified the integrity of any existing data, you can remove the backup data directory:

```
$ sudo rm -Rf /var/lib/mysql.bak
```

Restart MySQL one final time to be sure that it works as expected:

```
$ sudo systemctl restart mysql
$ sudo systemctl status mysql
```

Conclusion

In this tutorial, we've moved MySQL's data directory to a new location and updated Ubuntu's AppArmor ACLs to accommodate the adjustment. Although we were using a Block Storage device, the instructions here should be suitable for redefining the location of the data directory regardless of the underlying technology.

For more on managing MySQL's data directories, see these sections in the official MySQL documentation:

- The Mysql Data Directory
- Setting Up Multiple Data Directories

By: Melissa Anderson By: Hanif Jetha

Related Tutorials

How To Ensure Code Quality with SonarQube on Ubuntu 18.04

How To Sync and Share Your Files with Seafile on Ubuntu 18.04

How To Troubleshoot Issues in MySQL

How To Set Up a Remote Database to Optimize Site Performance with MySQL on Ubuntu 18.04

How To Set Up Laravel, Nginx, and MySQL with Docker Compose

3 Comments

Leave a comment...

Log In to Comment

shivamphp *November 20, 2018*

0 Hi There,

I am using Ubuntu 18.04 and I am trying to move my mysql data directory to external HDD.

I followed all the steps but at end when I tried to start the Mysql server it gives me below error.

==========

systemctl status mysql.service
● mysql.service - MySQL Community Server
Loaded: loaded (/lib/systemd/system/mysql.service; enabled; vendor preset: enabled)
Active: failed (Result: exit-code) since Tue 2018-11-20 10:30:52 EST; 4s ago
Process: 24879 ExecStart=/usr/sbin/mysqld --daemonize --pid-file=/run/mysqld/mysqld.pid (code=exited,
Process: 24869 ExecStartPre=/usr/share/mysql/mysql-systemd-start pre (code=exited, status=0/SUCCESS)
Main PID: 1481 (code=exited, status=0/SUCCESS)

Nov 20 10:30:52 Shivam systemd[1]: mysql.service: Service hold-off time over, scheduling restart.
Nov 20 10:30:52 Shivam systemd[1]: mysql.service: Scheduled restart job, restart counter is at 5.
Nov 20 10:30:52 Shivam systemd[1]: Stopped MySQL Community Server.
Nov 20 10:30:52 Shivam systemd[1]: mysql.service: Start request repeated too quickly.

Nov 20 10:30:52 Shivam systemd[1]: mysql.service: Failed with result 'exit-code'.
Nov 20 10:30:52 Shivam systemd[1]: Failed to start MySQL Community Server.

==========

Can you suggest me what is missing and how to fix this?

Thanks,
Shivam

---

anthonyd45e2ac81a9f79fc032 *December 27, 2018*

0 have used this tutorial many times before with success, but now running into issues on 18.04 with an external ssd mounted as:

```
/dev/sda1 on /data type ext4 (rw,nosuid,nodev,relatime,data=ordered,x-gvfs-show)
```

mysql works fine with the data dir in the default location (i've moved it out, broken it, and moved it back and it worked, just to be sure). i've triple-checked apparmor and i dont think that's the issue.

i think it has something to do with ownership or permissions on the parent folder (the /data mount point) since the mysql folder is just the rsync'ed one from the default location and all the qonership and permissions are intact.

here are the relevant permissions for the data directory.

```
drwx------   4 root root  4096 Dec 27 16:28 data
```

and the permissions inside /data

```
/data# ls -la
total 28
drwx------  4 root  root   4096 Dec 27 16:28 .
drwxr-xr-x 25 root  root   4096 Dec 27 16:22 ..
drwx------  2 root  root  16384 Dec 23 10:19 lost+found
drwx------  5 mysql mysql  4096 Dec 27 17:12 mysql
```

any suggestions appreciated. mysql fails to start with a 13 error trying to write the test temp files so im pretty sure this is what's what. just not sure how to fix.

---

gwijaya *January 23, 2019*

0 What if your 'data' directory permission is 755? ('only the 'data'. Leave the 'mysql' as 700)