Mariabackup + Restic: A simple and efficient online backup solution for your DBs

Faustin Lammler faustin@mariadb.org
Menu

- Presentation of MariaDB-backup
  - Normal backup
  - Incremental backup
- Presentation of the Restic backup solution
  - Normal backup
  - Incremental backup
- Restic remote storage (backends)
- Performance considerations
- Tips
- Links
Mariabackup

Mariabackup or mariadb-backup is an open source tool provided by MariaDB for performing **physical online** backups of InnoDB, Aria and MyISAM tables. For InnoDB, "hot online" backups are possible.

It was originally forked from Percona XtraBackup 2.3.8. It is available on Linux and Windows, see: https://mariadb.com/kb/en/mariabackup-overview/.

Note: you can use `mariabackup` or `mariadb-backup` command, the first one being a symbolic link to the second one.
Installation

The **mariadb-backup** executable is included in binary tarballs on Linux: [https://mariadb.com/kb/en/installing-mariadb-binary-tarballs/](https://mariadb.com/kb/en/installing-mariadb-binary-tarballs/)

Via the package manager:

- **Debian/Ubuntu**

  ```bash
  sudo apt-get install mariadb-backup
  ```

- **RedHat/Fedora**

  ```bash
  sudo dnf install MariaDB-backup
  ```
Creating local backup (1)

# mariadb-backup --user=root --backup --target-dir=/data/backup_db

[00] 2022-01-17 10:49:29 Connecting to MySQL server host: localhost
[00] 2022-01-17 10:49:29 Using server version 10.5.13-MariaDB-1:10.5.13+maria-buster-log
mariadb-backup based on MariaDB server 10.5.13-MariaDB debian-linux-gnu (x86_64)
[00] 2022-01-17 10:49:29 uses posix_fadvise().
[00] 2022-01-17 10:49:29 cd to /var/lib/mysql/
...
[00] 2022-01-17 10:52:23 Writing backup-my.cnf
[00] 2022-01-17 10:52:23 ...done
[00] 2022-01-17 10:52:23 Writing xtrabackup_info
[00] 2022-01-17 10:52:23 ...done
[00] 2022-01-17 10:52:23 Redo log (from LSN 1462416712885 to 1462442558594) was copied.
[00] 2022-01-17 10:52:23 completed OK!
Creating local backup (2)

```bash
# tree -L 1 /data/backup_db
/data/backup_db
    ├── aria_log.000000001
    ├── aria_log_control
    ├── backup-my.cnf
    ├── db1
    ├── db2
    ├── ib_buffer_pool
    ├── ibdata1
    ├── ib_logfile0
    ├── mysql
    ├── performance_schema
    ├── xtrabackup_binlog_info
    ├── xtrabackup_checkpoints
    └── xtrabackup_info
```
Creating incremental local backup (1)

First incremental backup (from `/data/backup_db`):

```
# mariadb-backup --user=root --backup \
--target-dir=/data/backup_db1 \
--incremental-basedir=/data/backup_db
```

Next incremental backup (from `/data/backup_db1`):

```
# mariadb-backup --user=root --backup \
--target-dir=/data/backup_db2 \ 
--incremental-basedir=/data/backup_db1
```
Creating incremental local backup (2)

And so on:

```bash
# tree -L 1 /data
/data
  ├── backup_db
  │     ├── backup_db1
  │     └── backup_db2
  │         ├── backup_db3
  │         └── backup_db4
  ⋯
  └── backup_dbX
```
Restoring full backups (1)

1. prepare the backup

```
# mariadb-backup --prepare --target-dir=/data/backup_db
```

2. stop MariaDB server

```
# systemctl stop mariadb
```

3. remove datadir

```
# mv /var/lib/mysql/ /var/lib/mysql_old
# mkdir /var/lib/mysql
```
Restoring full backups (2)

4. move (\texttt{--move-back}) or copy (\texttt{--copy-back}) from backup

\begin{verbatim}
# mariadb-backup --move-back --target-dir=/data/backup_db
\end{verbatim}

5. fix rights

\begin{verbatim}
# chown -R mysql:mysql /var/lib/mysql
\end{verbatim}

6. start MariaDB server

\begin{verbatim}
# systemctl start mariadb
\end{verbatim}
Restoring incrementals backups

The preparation step (1) needs an extra task to include all the incremental backups.

1. prepare the first full backup and include all increments (the order of incremental backups inclusion is important):

```
# mariadb-backup --prepare --target-dir=/data/backup_db
```

```
for dir in /data/backup_db1 /data/backup_db2 /data/backup_db3; do
    mariadb-backup --prepare --target-dir=/data/backup_db \
    --incremental-dir="$dir"
done
```

Then you can apply the exact same step as for a normal backup (2, 3, 4, 5 and 6).
The **--stream=xbstream** option (1)

There are plenty of options available (**--compress**, **--encrypted-backup**, etc.), see: `mariadb-backup --help`. But instead of using those, it's recommended to use dedicated compression or encryption tools with the **--stream=xbstream** option.

For instance, you can compress the backup on the fly with gzip:

```
# mariadb-backup --user=root --backup --stream=xbstream | gzip >mariadb.xb.gz
```

Uncompress:

```
gunzip -c mariadb.xb.gz | mbstream -x
```
The **--stream=xbstream** option (2)

The same goes for encryption:

```
mariadb-backup --user=root --backup --stream=xbstream |
gpg -c --passphrase SECRET --batch --yes -o mariadb.xb.gpg
```

Decrypt:

```
gpg --decrypt --passphrase SECRET --batch --yes mariadb.xb.gpg |
mbstream -x
```
Restic

Written in Go, Restic is a fast and secure backup program that supports a wide range of storages (backends) [https://restic.readthedocs.io/](https://restic.readthedocs.io/).

By coupling mariadb-backup with it, it becomes very easy to implement:

- incremental backups;
- remote storage;
- encryption.
Installation

You can download the latest stable release versions of Restic from the Restic release page (https://github.com/restic/restic/releases/latest).

Note that there is a very useful `self-update` option:

`sudo restic self-update`

Via the package manager:

- **Debian/Ubuntu**
  
  `sudo apt-get install restic`

- **RedHat/Fedora**
  
  `sudo dnf install restic`
Local backup with Restic (1)

1. initiate the Restic repository:

```bash
# export RESTIC_REPOSITORY="/data/backup_restic"
# export RESTIC_PASSWORD="mypassword1243876123!!-__"
# restic init
created restic repository e45984e7 at /data/backup_restic
```

Please note that knowledge of your password is required to access the repository. Losing your password means that your data is irrecoverably lost.
Local backup with Restic (2)

Restic repository tree:

```
# tree -L 1 backup_restic/
backup_restic/
    ├── config
    ├── data
    ├── index
    ├── keys
    ├── locks
    └── snapshots

5 directories, 1 file
```
2. create the backup

```bash
# mariadb-backup --user=root --backup --stream=xbstream 2>/data/mariadb-backup.log | restic backup --stdin --stdin-filename mariadb.xb --tag MariaDB

repository e45984e7 opened successfully, password is correct
created new cache in /root/.cache/restic
no parent snapshot found, will read all files
Files:           1 new,     0 changed,     0 unmodified
Dirs:            0 new,     0 changed,     0 unmodified
Added to the repo: 61.762 GiB

processed 1 files, 63.062 GiB in 3:35
snapshot 6b2acceb saved
```

```bash
# restic snapshots
repository e45984e7 opened successfully, password is correct
ID        Time                 Host        Tags        Paths
------------------------------------------------------------------
6b2acceb  2022-01-18 12:52:16  hz-bbm1     MariaDB     /mariadb.xb
------------------------------------------------------------------
1 snapshots
```
Local backup with Restic (4)

3. incremental backup (same command as above!):

```
# mariadb-backup --user=root --backup --stream=xbstream 2>/data/mariadb-backup.log | restic backup --stdin --stdin-filename mariadb.xb --tag MariaDB
repository e45984e7 opened successfully, password is correct
using parent snapshot 6b2acce6
Files: 0 new, 1 changed, 0 unmodified
Dirs: 0 new, 0 changed, 0 unmodified
Added to the repo: 2.156 GiB
processed 1 files, 63.017 GiB in 2:03
snapshot 718e9e02 saved

# restic snapshots
repository e45984e7 opened successfully, password is correct
ID     Time            Host     Tags     Paths
------- --------------- ---------- ------- ----------
6b2acce6 2022-01-18 12:52:16  hz-bbm1  MariaDB /mariadb.xb
718e9e02 2022-01-18 12:58:44  hz-bbm1  MariaDB /mariadb.xb

2 snapshots
```

Easy!
Error handling

Since `--stream` option streams backup files to STDOUT, the mariadb-backup log is sent to STDERR. This is intentional (see: https://jira.percona.com/browse/PXB-1469).

That is why the use of `STDERR (2>/data/mariadb-backup.log)` in the previous command should be taken in consideration at the moment of handling eventual backup errors.
**Restore Local backup**

Use `restic snapshots` to determine which snapshot you want to restore or use the `latest` keyword (see: [https://restic.readthedocs.io/en/stable/050_restore.html](https://restic.readthedocs.io/en/stable/050_restore.html)).

```bash
# restic restore latest --target .
```

Un-serialize the backup with `mbstream`:

```bash
# mkdir mariadb_recovery && cd mariadb_recovery
# mbstream -x <../mariadb.xb
```

Prepare the recovery:

```bash
# mariadb-backup --prepare --target-dir=.
```
What about remote storage (Restic backends)?

**SFTP**

```
# export RESTIC_REPOSITORY="sftp:100.64.200.20:/data/restic_backup"
```

**S3 (AWS or Minio Server)**


```
# export AWS_ACCESS_KEY_ID="AKIAJAJSLTZCAZ4SRI5Q"
# export AWS_SECRET_ACCESS_KEY="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"
# export RESTIC_REPOSITORY="s3:https://s3.amazonaws.com/restic-demo"
```

See the full list of available backends:
Performance (1)

It's difficult to benchmark correctly mariadb-backup vs mariadb-backup + Restic. This depends on various parameters:

- the load on the DB at the moment of the backup (delta of increments);
- network load and the bandwidth availability in the case of a remote storage backup;
- IO disk available (if local storage);
- Restic adds various overheads (encryption, de-duplication, indexing, see https://restic.readthedocs.io/en/latest/100_references.html).
Performance (2)

The following array shows a quick comparison of a local backup with mariadb-backup and with mariadb-backup + Restic (DB backup size ~64GB):

<table>
<thead>
<tr>
<th>type</th>
<th>time first backup</th>
<th>time incr backup (*)</th>
</tr>
</thead>
<tbody>
<tr>
<td>mariadb-backup</td>
<td>119 s</td>
<td>44 s</td>
</tr>
<tr>
<td>mariadb-backup + Restic</td>
<td>219 s</td>
<td>132 s</td>
</tr>
</tbody>
</table>

(*): after same amount of time (DB is updated regularly)
Check your backups!

Example with Podman (or Docker).

In a terminal execute the following commands (mariadb_recovery contains the restored datadir):

```
# podman run -it -v ./mariadb_recovery:/var/lib/mysql docker.io/mariadb:10.5 bash
root@6d5b6f121bb9:/# chown -R mysql:mysql /var/lib/mysql
```

In a second terminal execute the following commands to start the mariadb server (container id is the previous one):

```
# podman exec -it -u mysql 6d5b6f121bb9 bash
mysql@6d5b6f121bb9:/$ export MARIADB_ROOT_PASSWORD="recover"
mysql@6d5b6f121bb9:/$ /usr/local/bin/docker-entrypoint.sh mysqld
```

You can then connect to the DB from the first terminal and verify that the database contains everything.
Tips

Automate recovery and check process: why not injecting your backups from Restic snapshots in your staging environment every night?

Tag your backups snapshots (example `--tag MariaDB`), this will help future retention strategies.

Use the `nice` command if Restic takes too many resources (encryption, de-duplication and indexing overhead):

```bash
mariadb-backup ... | nice -n19 restic backup --stdin ...
```

Now that you know how to easily and quickly restore a full DB from scratch with your backups, why not creating a secondary node and handle your backups from that node. Note that contrary to mariadb-dump (logical backup), there is no lock during the backup if you use InnoDB so this is less important.
Links

Mariabackup doc:
https://mariadb.com/kb/en/mariabackup/

Report mariadb-backup issues:
https://jira.mariadb.org

Restic doc:
https://restic.readthedocs.io/

Restic cryptography analysis (by Filippo Valsorda):
https://blog.filippo.io/restic-cryptography/

Use Borg instead of Restic (not tested)?
Questions?

- faustin@mariadb.org
- https://mariadb.org/about/#faustin-lammler
Sponsors

- Tencent Cloud
- Microsoft
- MariaDB
- servicenow
- Alibaba Cloud
- VISMA
- IBM
- DBS
- AUTOMATIC