Instalação e Customização MariaDB Docker

- Mariabackup tutorial — Using Mariabackup for hot physical backups of MariaDB

Mariabackup tutorial — Using Mariabackup for hot physical backups of MariaDB

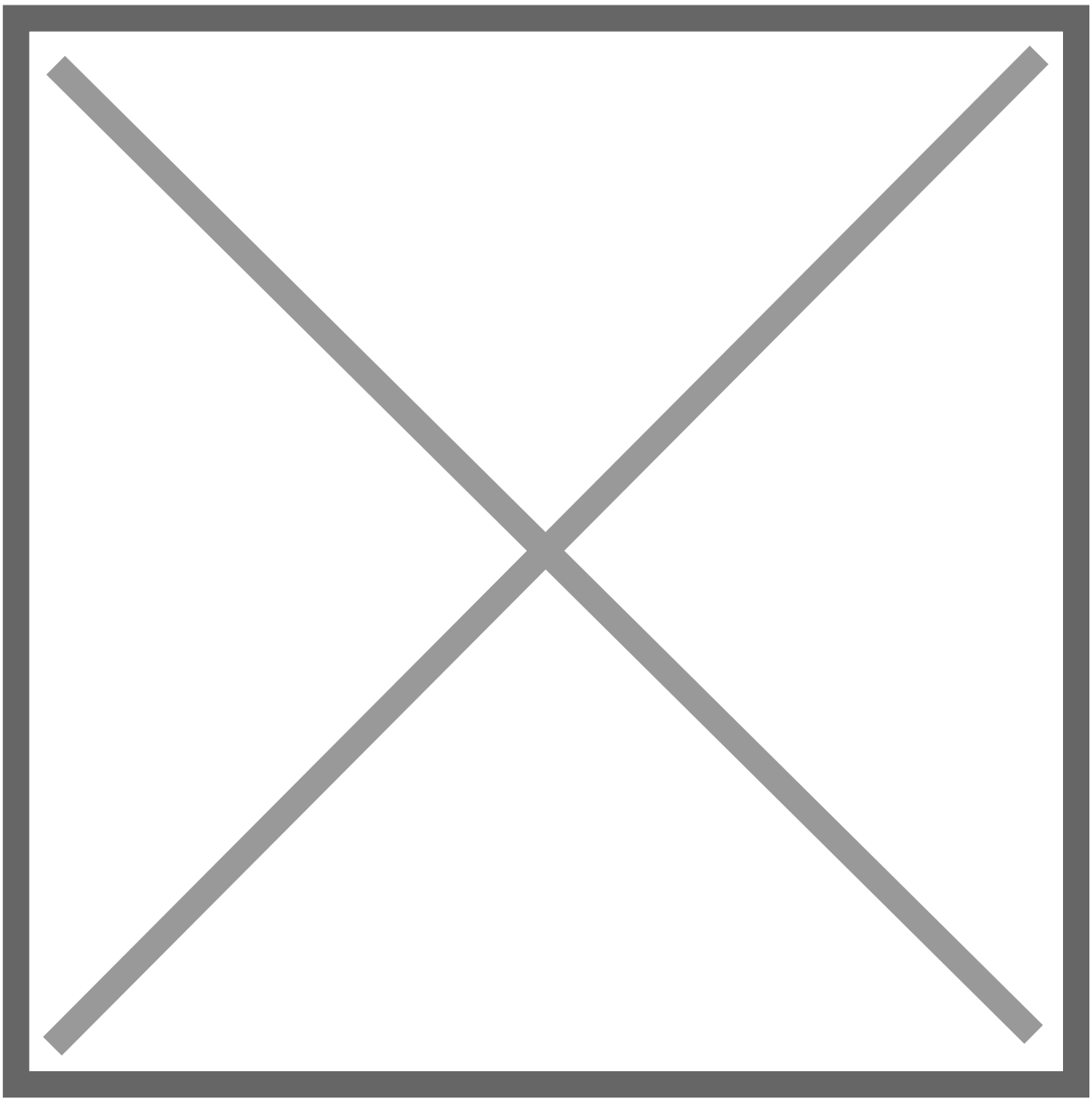
Link:

https://medium.com/@ngza5tqf/mariabackup-tutorial-using-mariabackup-for-hot-physical-backups-of-mariadb-c5106b9dd861

Jan 18, 2026

Mariabackup is MariaDB's native tool for creating hot physical backups without interrupting database operations. Unlike logical backups with mysqldump, Mariabackup copies the actual data files while your database continues serving requests. This makes it the preferred backup method for production MariaDB databases where downtime is not an option. This tutorial covers everything from basic usage to advanced incremental backup strategies for MariaDB backup operations.

Press enter or click to view image in full size



Mariabackup tutorial

Understanding Mariabackup

Mariabackup is a fork of Percona XtraBackup, specifically optimized for MariaDB. It creates consistent physical backups by copying InnoDB data files and capturing transaction logs during the backup process. The tool then applies these logs during a preparation phase to ensure backup consistency.

How Mariabackup works

The backup process happens in three distinct phases. First, Mariabackup copies InnoDB tablespace files while the database runs normally. During this copy phase, it also captures all changes happening in the redo log. After the file copy completes, Mariabackup briefly acquires a global lock to capture non-InnoDB tables and the binary log position. Finally, during the prepare phase, Mariabackup applies the captured redo log entries to make the backup consistent.

This approach allows backups to complete without blocking database operations. Applications continue reading and writing data throughout most of the backup process. The brief lock at the end typically lasts only seconds, even for large databases.

Mariabackup vs mysqldump

Both tools create valid backups, but they serve different use cases.

Mariabackup characteristics:

- **Backup type**: Physical (file copy)
- **Hot backup**: Yes
- **Backup speed**: Fast
- **Restore speed**: Fast
- **Incremental backups**: Yes
- **Point-in-time recovery**: Yes
- **Cross-version restore**: Same major version only
- **Storage size**: Larger (raw files)

mysqldump characteristics:

- **Backup type**: Logical (SQL statements)
- **Hot backup**: Partial (InnoDB only with — single-transaction)
- **Backup speed**: Slow for large databases
- **Restore speed**: Slow (executes SQL statements)
- **Incremental backups**: No
- **Point-in-time recovery**: Yes (with binary logs)
- **Cross-version restore**: Any compatible version
- **Storage size**: Smaller (compressed SQL)

For databases under 10GB, mysqldump works fine and offers better portability. For larger databases or systems requiring minimal downtime, Mariabackup is the better choice. Many production environments use both: Mariabackup for fast daily backups and mysqldump for portable weekly archives.

Installing Mariabackup

Mariabackup comes bundled with MariaDB Server packages. If you have MariaDB installed, you likely already have Mariabackup available.

Installation on Debian/Ubuntu

Install the mariadb-backup package:

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

Verify the installation:

```
mariabackup --version
```

Installation on RHEL/CentOS

Use yum or dnf to install:

```
sudo yum install MariaDB-backup
```

Or on newer systems:

```
sudo dnf install MariaDB-backup
```

Installation on Docker

MariaDB Docker images include Mariabackup. Access it through docker exec:

```
docker exec mariadb mariabackup --version
```

For Docker-based backups, mount a volume for backup storage:

```
docker exec mariadb mariabackup --backup \
  --target-dir=/backup \
  --user=root \
```

```
  --password=your_password
```

Version compatibility

Mariabackup version must match your MariaDB server version. A Mariabackup from MariaDB 10.6 cannot backup a MariaDB 10.11 server. Always use the Mariabackup that ships with your MariaDB installation.

Check your MariaDB version:

```
mariadb --version
```

Ensure Mariabackup matches:

```
mariabackup --version
```

Creating your first backup

The basic Mariabackup command requires a target directory, username and password. The backup process creates a complete copy of your database files.

Basic backup command

Create a full backup:

```
mariabackup --backup \
  --target-dir=/backup/full-$(date +%Y%m%d) \
  --user=backup_user \
  --password=your_password
```

The `--target-dir` must be an empty directory or non-existent (Mariabackup creates it). Never reuse a target directory from a previous backup.

Backup user permissions

Create a dedicated backup user with minimal required privileges:

```
CREATE USER 'backup_user'@'localhost' IDENTIFIED BY 'secure_password';
GRANT RELOAD, PROCESS, LOCK TABLES, REPLICATION CLIENT ON *.* TO 'backup_user'@'localhost';
FLUSH PRIVILEGES;
```

For MariaDB 10.5 and later, you can use the BACKUP_ADMIN privilege instead:

```
CREATE USER 'backup_user'@'localhost' IDENTIFIED BY 'secure_password';
GRANT RELOAD, PROCESS, LOCK TABLES, BINLOG MONITOR, BACKUP_ADMIN ON *.* TO 'backup_user'@'localh
FLUSH PRIVILEGES;
```

Backup directory structure

After a successful backup, your target directory contains:

```
/backup/full-20260118/
├── ibdata1
├── ib_logfile0
├── ib_logfile1
├── mysql/
├── performance_schema/
├── your_database/
├── xtrabackup_checkpoints
├── xtrabackup_info
└── xtrabackup_logfile
```

The `xtrabackup_checkpoints` file contains backup metadata including the LSN (Log Sequence Number) range. This information is essential for incremental backups.

Preparing backups for restoration

Raw Mariabackup output is not directly restorable. The prepare phase applies redo log entries to make data files consistent. This step is mandatory before restoration.

Preparing a full backup

Run the prepare command on your backup directory:

```
mariabackup --prepare --target-dir=/backup/full-20260118
```

During preparation, Mariabackup:

1. Reads the captured redo log entries
2. Applies committed transactions to data files
3. Rolls back uncommitted transactions
4. Creates clean, consistent data files

The prepare phase modifies the backup directory in place. After preparation, the backup is ready for restoration but can no longer be used as a base for incremental backups.

Preparation output

A successful prepare shows:

```
InnoDB: Starting shutdown...
InnoDB: Shutdown completed
completed OK!
```

If you see errors during preparation, the backup may be corrupted. Never restore from a backup that fails preparation.

Restoring from backup

Restoration requires stopping MariaDB, replacing the data directory and fixing permissions. Always test restoration procedures before you need them in an emergency.

Full restoration procedure

Stop the MariaDB service:

```
sudo systemctl stop mariadb
```

Remove or move the existing data directory:

```
sudo mv /var/lib/mysql /var/lib/mysql.old
sudo mkdir /var/lib/mysql
```

Copy the backup files to the data directory:

```
sudo mariabackup --copy-back --target-dir=/backup/full-20260118
```

Fix file ownership:

```
sudo chown -R mysql:mysql /var/lib/mysql
```

Start MariaDB:

```
sudo systemctl start mariadb
```

Alternative: move-back option

If you don’t need to preserve the backup, use `--move-back` instead of `--copy-back`:

```
sudo mariabackup --move-back --target-dir=/backup/full-20260118
```

This moves files instead of copying, which is faster and uses less disk space. However, your backup directory becomes empty after this operation.

Verifying restoration

Connect to MariaDB and verify your data:

```
mariadb -u root -p -e "SHOW DATABASES;"
mariadb -u root -p -e "SELECT COUNT(*) FROM your_database.your_table;"
```

Check the error log for any issues:

```
sudo tail -100 /var/log/mysql/error.log
```

Incremental backups

Incremental backups capture only changes since the last backup, reducing backup time and storage requirements. They work by recording the LSN at each backup and only copying pages modified after that LSN.

Creating an incremental backup

First, create a full backup as the base:

```
mariabackup --backup \
  --target-dir=/backup/base \
  --user=backup_user \
  --password=your_password
```

Create the first incremental backup:

```
mariabackup --backup \
  --target-dir=/backup/inc1 \
  --incremental-basedir=/backup/base \
  --user=backup_user \
  --password=your_password
```

Create subsequent incremental backups based on the previous one:

```
mariabackup --backup \
  --target-dir=/backup/inc2 \
  --incremental-basedir=/backup/inc1 \
  --user=backup_user \
  --password=your_password
```

Preparing incremental backups

Incremental backups require a multi-step preparation process. First, prepare the base backup with the `--apply-log-only` option:

```
mariabackup --prepare --apply-log-only --target-dir=/backup/base
```

The `--apply-log-only` flag prevents the rollback phase, which would make the backup unusable for applying incremental changes.

Get Nazar Egorov’s stories in your inbox

Join Medium for free to get updates from this writer.

☑

Remember me for faster sign in

Apply the first incremental backup:

```
mariabackup --prepare --apply-log-only \
  --target-dir=/backup/base \
  --incremental-dir=/backup/inc1
```

Apply the second incremental backup:

```
mariabackup --prepare --apply-log-only \
  --target-dir=/backup/base \
  --incremental-dir=/backup/inc2
```

For the final incremental backup, run prepare without `--apply-log-only`:

```
mariabackup --prepare --target-dir=/backup/base
```

This final preparation completes the rollback phase and makes the backup restorable.

Incremental backup strategy

A common production strategy combines weekly full backups with daily incremental backups:

- **Sunday**: Full backup
- **Monday-Saturday**: Incremental backups based on previous day

This approach provides daily recovery points while minimizing storage and backup time. Recovery requires applying all incremental backups in sequence, so keep the chain reasonably short to limit restore complexity.

Streaming backups

Mariabackup supports streaming backup output directly to another server or compression tool. This eliminates the need for temporary local storage.

Streaming to a file

Create a compressed backup archive:

```
mariabackup --backup \
  --stream=xbstream \
  --user=backup_user \
  --password=your_password | gzip > /backup/backup_$(date +%Y%m%d).xbstream.gz
```

Streaming to a remote server

Send backups directly to a remote server via SSH:

```
mariabackup --backup \
  --stream=xbstream \
  --user=backup_user \
  --password=your_password | \
  ssh backup-server "cat > /backup/backup_$(date +%Y%m%d).xbstream"
```

Extracting streamed backups

Extract the xbstream archive before preparation:

```
mkdir /backup/extracted
cd /backup/extracted
xbstream -x < /backup/backup_20260118.xbstream.gz
```

Then prepare and restore as usual.

Partial backups

Mariabackup can backup specific databases or tables, useful when you only need to protect certain data or have limited storage.

Backing up specific databases

Use the `--databases` option to backup selected databases:

```
mariabackup --backup \
  --target-dir=/backup/partial \
  --databases="production inventory" \
```

```
  --user=backup_user \
  --password=your_password
```

Backing up specific tables

Backup individual tables with the `--tables` option:

```
mariabackup --backup \
  --target-dir=/backup/tables \
  --tables="production.orders production.customers" \
  --user=backup_user \
  --password=your_password
```

Partial backup limitations

Partial backups have restrictions:

- Cannot restore to a running server (requires full data directory replacement)
- InnoDB system tablespace (ibdata1) is always included
- Foreign key relationships may cause issues if related tables are excluded

For most use cases, full backups with selective restoration provide more flexibility than partial backups.

Encrypted backups

Mariabackup supports AES encryption to protect backup files at rest. Encryption happens during the backup process, so unencrypted data never touches disk.

Creating encrypted backups

Generate an encryption key:

```
openssl rand -base64 32 > /etc/mysql/backup.key
chmod 600 /etc/mysql/backup.key
```

Create an encrypted backup:

```
mariabackup --backup \
  --target-dir=/backup/encrypted \
  --encrypt=AES256 \
  --encrypt-key-file=/etc/mysql/backup.key \
  --user=backup_user \
  --password=your_password
```

Decrypting backups

Decrypt before preparation:

```
mariabackup --decrypt=AES256 \
  --encrypt-key-file=/etc/mysql/backup.key \
  --target-dir=/backup/encrypted
```

Then prepare and restore normally.

Key management

Store encryption keys separately from backup files. If an attacker gains access to your backups but not the keys, your data remains protected. Consider using key management services like HashiCorp Vault or AWS KMS for production environments.

Automated backups with Databasus

Manual backup scripts work for simple setups, but production environments benefit from dedicated backup management tools. Databasus is a free, open source backup solution that automates MariaDB backups with scheduling, multiple storage destinations and team notifications.

Installing Databasus

Install Databasus using Docker:

```
docker run -d \
  --name databasus \
```

```
  -p 4005:4005 \
  -v ./databasus-data:/databasus-data \
  --restart unless-stopped \
  databasus/databasus:latest
```

Or with Docker Compose:

```
services:
  databasus:
    container_name: databasus
    image: databasus/databasus:latest
    ports:
      - "4005:4005"
    volumes:
      - ./databasus-data:/databasus-data
    restart: unless-stopped
```

Start the service:

```
docker compose up -d
```

Access the web interface at `http://localhost:4005` and create your account.

Configuring MariaDB backups

1. **Add your database**: Click “New Database” and select MariaDB as the database type
2. **Enter connection details**: Provide your MariaDB host, port, database name and authentication credentials
3. **Select storage**: Choose from local storage, S3, Google Cloud Storage, Dropbox, SFTP or other supported destinations
4. **Configure schedule**: Set hourly, daily, weekly, monthly or custom cron-based backup intervals
5. **Add notifications** (optional): Configure Slack, Discord, Telegram or email alerts for backup status
6. **Create backup**: Databasus validates your configuration and begins the backup schedule

Databasus handles compression, encryption and retention automatically, removing the complexity of backup management while providing enterprise-grade reliability.

Backup automation scripts

For environments where you prefer script-based automation, here’s a production-ready backup script.

Full backup script

Create `/usr/local/bin/mariadb-backup.sh`:

```
#!/bin/bash
set -e
```

```
BACKUP_BASE="/backup/mariadb"
DATE=$(date +%Y%m%d_%H%M%S)
BACKUP_DIR="$BACKUP_BASE/full-$DATE"
LOG_FILE="/var/log/mariadb-backup.log"
RETENTION_DAYS=7log() {
    echo "[$(date '+%Y-%m-%d %H:%M:%S')] $1" >> "$LOG_FILE"
}log "Starting full backup to $BACKUP_DIR"mariabackup --backup \
  --target-dir="$BACKUP_DIR" \
  --user=backup_user \
  --password="$(cat /etc/mysql/backup.password)" \
  2>> "$LOG_FILE"log "Preparing backup"mariabackup --prepare --target-dir="$BACKUP_DIR" 2>> "$LO
find "$BACKUP_BASE" -maxdepth 1 -type d -name "full-*" -mtime +$RETENTION_DAYS -exec rm -rf {} \
```

Make it executable and schedule with cron:

```
chmod +x /usr/local/bin/mariadb-backup.sh
```

Add to crontab for daily 2 AM backups:

```
0 2 * * * /usr/local/bin/mariadb-backup.sh
```

Incremental backup script

Create `/usr/local/bin/mariadb-incremental.sh`:

```
#!/bin/bash
set -e
```

```
BACKUP_BASE="/backup/mariadb"
DATE=$(date +%Y%m%d_%H%M%S)
DAY_OF_WEEK=$(date +%u)
LOG_FILE="/var/log/mariadb-backup.log"log() {
    echo "[$(date '+%Y-%m-%d %H:%M:%S')] $1" >> "$LOG_FILE"
```

```
}# Sunday (day 7) = full backup, other days = incremental
if [ "$DAY_OF_WEEK" -eq 7 ]; then
    BACKUP_DIR="$BACKUP_BASE/base"
    rm -rf "$BACKUP_DIR"

    log "Starting weekly full backup"

    mariabackup --backup \
      --target-dir="$BACKUP_DIR" \
      --user=backup_user \
      --password="$(cat /etc/mysql/backup.password)" \
      2>> "$LOG_FILE"
else
    # Find the most recent backup directory
    LAST_BACKUP=$(ls -td "$BACKUP_BASE"/*/ 2>/dev/null | head -1)
    BACKUP_DIR="$BACKUP_BASE/inc-$DATE"

    log "Starting incremental backup based on $LAST_BACKUP"

    mariabackup --backup \
      --target-dir="$BACKUP_DIR" \
      --incremental-basedir="$LAST_BACKUP" \
      --user=backup_user \
      --password="$(cat /etc/mysql/backup.password)" \
      2>> "$LOG_FILE"
filog "Backup completed: $BACKUP_DIR"
```

Monitoring and verification

Backups are worthless if they don’t restore. Regular verification catches problems before they become emergencies.

Backup verification script

Create a script that tests restoration to a temporary instance:

```
#!/bin/bash
set -e
```

```
BACKUP_DIR="/backup/mariadb/full-latest"
TEST_DATA_DIR="/tmp/mariadb-test"
TEST_PORT=3307# Clean up previous test
rm -rf "$TEST_DATA_DIR"
mkdir -p "$TEST_DATA_DIR"# Copy backup to test directory
mariabackup --copy-back --target-dir="$BACKUP_DIR" --datadir="$TEST_DATA_DIR"# Fix permissions
```

```
chown -R mysql:mysql "$TEST_DATA_DIR"# Start test instance
mariadbd --datadir="$TEST_DATA_DIR" --port=$TEST_PORT --socket=/tmp/mysql-test.sock &
TEST_PID=$!sleep 10# Verify databases exist
mariadb -P $TEST_PORT -S /tmp/mysql-test.sock -e "SHOW DATABASES;"# Run table checks
mariadb -P $TEST_PORT -S /tmp/mysql-test.sock -e "CHECK TABLE production.orders;"# Cleanup
kill $TEST_PID
rm -rf "$TEST_DATA_DIR"echo "Backup verification completed successfully"
```

Monitoring backup health

Track these metrics to ensure backup reliability:

- **Backup duration**: Sudden increases indicate performance issues
- **Backup size**: Unexpected changes may signal data problems
- **Last successful backup**: Alert if older than expected
- **Preparation success**: Failed preparation means unusable backup

Set up alerts for backup failures. A missed backup is better discovered immediately than during a crisis.

Troubleshooting common issues

Even well-configured backups encounter problems. Here are solutions to common Mariabackup issues.

Lock wait timeout

Error: `Lock wait timeout exceeded`

This happens when Mariabackup cannot acquire the global lock. Reduce lock time by:

- Running backups during low-traffic periods
- Killing long-running transactions before backup
- Using `--lock-ddl-per-table` option (MariaDB 10.4+)

Insufficient disk space

Error: `No space left on device`

Mariabackup requires free space equal to your database size plus redo logs. Solutions:

- Use streaming backups to compress on-the-fly
- Clean up old backups before starting new ones
- Use a dedicated backup volume with adequate space

Redo log overwritten

Error: `The log was only partially backed up`

The redo log filled and wrapped around during backup. Fix by:

- Increasing `innodb_log_file_size` in MariaDB configuration
- Running backups more frequently
- Reducing database write load during backups

Permission denied errors

Error: `Permission denied`

Mariabackup needs read access to data files and write access to the target directory. Verify:

- Backup user has correct MySQL privileges
- System user running Mariabackup can read `/var/lib/mysql`
- Target directory is writable

Best practices

Following these practices ensures reliable, maintainable backup operations.

The 3–2–1 backup rule

Implement the 3–2–1 strategy:

- **3 copies** of your data: production database, local backup and remote backup
- **2 different media types**: local disk and cloud storage
- **1 offsite location**: different physical location from production

This protects against hardware failures, site disasters and ransomware attacks.

Test restoration regularly

Schedule monthly restoration tests:

1. Select a random backup from the previous week
2. Restore to a test environment
3. Verify data integrity and completeness
4. Document restoration time
5. Update runbooks based on findings

Document everything

Create runbooks covering:

- Backup schedules and retention policies
- Step-by-step restoration procedures
- Contact information for escalations
- Recovery time objectives and test results

Train multiple team members on backup and restoration. Single points of failure in knowledge are as dangerous as single points of failure in infrastructure.

Secure your backups

Backups contain all your production data. Protect them accordingly:

- Encrypt backups at rest
- Restrict access to backup storage

- Audit backup access logs
- Store encryption keys separately from backups

Conclusion

Mariabackup provides fast, reliable hot backups for MariaDB databases of any size. Its ability to backup without blocking operations makes it essential for production systems where downtime costs money. Start with simple full backups, add incremental backups as your database grows and implement encryption for sensitive data.

Remember that creating backups is only half the job. Regular restoration testing, monitoring and documentation complete a reliable backup strategy. When disaster strikes, your preparation determines whether recovery takes minutes or becomes a crisis.

Database

Mariadb

Nazar Egorov

Written by Nazar Egorov

13 followers

·1 following

No responses yet



Write a response

What are your thoughts?

More from Nazar Egorov

PostgreSQL backup verification—How to test and validate your PostgreSQL backups
Nazar Egorov

Nazar Egorov

PostgreSQL backup verification — How to test and validate your PostgreSQL backups

A backup that can’t be restored is not a backup. Many teams discover their backups are corrupted, incomplete or otherwise unusable only…

Jan 20

1
MariaDB Docker setup—Running MariaDB in Docker containers complete guide
Nazar Egorov

Nazar Egorov

MariaDB Docker setup — Running MariaDB in Docker containers complete guide

Running MariaDB in Docker simplifies deployment, makes environments reproducible, and allows you to spin up databases in seconds. Whether…

Feb 1

PostgreSQL WAL archiving explained—Understanding Write-Ahead Logs for backup and recovery
Nazar Egorov

Nazar Egorov

PostgreSQL WAL archiving explained — Understanding Write-Ahead Logs for backup and recovery

Write-Ahead Logs are PostgreSQL’s insurance policy against data loss. Every transaction gets written to WAL before it touches your actual...

Jan 14

18

PostgreSQL backup encryption — How to encrypt your PostgreSQL database backups

Nazar Egorov

Nazar Egorov

PostgreSQL backup encryption — How to encrypt your PostgreSQL database backups

Unencrypted backups are a security liability. A database backup contains everything — user credentials, payment details, personal...

Jan 21

See all from Nazar Egorov

Recommended from Medium

PostgreSQL Backup Strategies
PostgreSQL Blogs

PostgreSQL Backup Strategies

Database engineers are divided into two types: those who take backups, and those who haven’t lost data yet.” We all know this old saying…

Jan 2

32
Mastering PostgreSQL Autovacuum: Tuning for Peak Performance
Borhadeshardul

Borhadeshardul

Mastering PostgreSQL Autovacuum: Tuning for Peak

Performance

When it comes to PostgreSQL performance, autovacuum is one of the most critical and often misunderstood features. It silently keeps your…

Oct 22, 2025

4

1

WAL and Vacuum Monitoring: The Two Metrics That Predict Every Outage
Philip McClarence

Philip McClarence

WAL and Vacuum Monitoring: The Two Metrics That Predict Every Outage

Every PostgreSQL outage post-mortem I have read — and I have read a lot of them — eventually traces back to one of two things: the disk…

Mar 7

1

PostgreSQL backup verification — How to test and validate your PostgreSQL backups
Nazar Egorov

Nazar Egorov

PostgreSQL backup verification — How to test and validate your PostgreSQL backups

A backup that can’t be restored is not a backup. Many teams discover their backups are corrupted, incomplete or otherwise unusable only…

Jan 20

1

PostgreSQL Performance Tuning That Actually Matters
Kevin Gabeci

Kevin Gabeci

PostgreSQL Performance Tuning That Actually Matters

The 20% of config changes that solve 80% of performance problems.

Mar 3

53

1

Customizing storage paths in Laravel / Lumen

ucama