

Preventing Full Disk Issues

Running out of disk space in a MySQL environment can result in failed writes, temporary unavailability, and even data corruption. MySQL requires space not only for storing table data and indexes, but also for binary logs, temporary tables, transaction logs, and background operations. On platforms like Elestio, while the infrastructure is managed, users are responsible for monitoring data growth, managing logs, and planning for scale. This guide covers how to monitor disk usage, configure alerts, clean up unused data, and follow best practices to prevent full disk scenarios in a MySQL setup.

Monitoring Disk Usage

Effective disk usage monitoring allows you to detect unexpected growth before it becomes critical. A combination of operating system-level checks and MySQL-specific queries gives a complete view of space consumption.

To inspect overall system storage from the terminal or container shell, use:

```
df -h
```

This command shows available and used space for each mount point. Identify the mount that hosts your MySQL data directory—usually `/var/lib/mysql` on Linux systems.

To check database-level usage inside MySQL, connect using the MySQL CLI and run:

```
SELECT table_schema AS db_name,  
       ROUND(SUM(data_length + index_length) / 1024 / 1024, 2) AS size_mb  
FROM information_schema.tables  
GROUP BY table_schema  
ORDER BY size_mb DESC;
```

This reveals the size of each database schema in megabytes, including both data and indexes. For insights at the table level, run:

```
SELECT table_name,  
       ROUND((data_length + index_length) / 1024 / 1024, 2) AS size_mb  
FROM information_schema.tables  
WHERE table_schema = 'your_database_name'
```

```
ORDER BY size_mb DESC  
LIMIT 10;
```

Replace 'your_database_name' with your actual schema name. This helps pinpoint which tables are growing fastest or consuming disproportionate space.

Configuring Alerts and Cleaning Up Storage

Monitoring alone isn't enough—automatic alerting and cleanup strategies ensure you're notified in time and can act without downtime. In Docker Compose setups, container disk usage can be reviewed using:

```
docker system df
```

This shows disk consumption across images, containers, and volumes. To list and inspect unused volumes:

```
docker volume ls
```

And to remove a specific unused volume:

```
docker volume rm <volume-name>
```

Do not remove any volume actively used by MySQL. Before any cleanup, confirm that your database volumes are backed up and not mounted by a running service. Within MySQL, temporary tables, binary logs, and undo logs can consume space rapidly. You can check the binary log directory and purge old logs manually:

```
SHOW BINARY LOGS;
```

To delete older binary logs and reclaim space:

```
PURGE BINARY LOGS BEFORE NOW() - INTERVAL 7 DAY;
```

This deletes logs older than 7 days. Adjust the interval based on your backup retention policy. You can also automate this behavior using the configuration option:

```
[mysqld]  
expire_logs_days = 7
```

Managing & Optimizing Temporary Files

MySQL uses temporary files for complex queries, especially those involving large sorts or joins without indexes. These files are stored in the tmpdir directory and can fill up if not managed. Monitor the temp directory using OS tools:

```
du -sh /tmp
```

If temp file usage is consistently high, consider tuning the tmp_table_size and max_heap_table_size variables to reduce reliance on disk-based temporary tables.

To identify tables with excessive unused space, use:

```
SHOW TABLE STATUS WHERE Data_free > 0;
```

These tables may benefit from optimization. Reclaim the unused space by running:

```
OPTIMIZE TABLE your_table_name;
```

This rewrites the table and defragments it, reclaiming disk space. For InnoDB tables, this can also compact the clustered index.

Best Practices for Disk Space Management

Long-term disk health in MySQL requires more than just cleanup—it demands strategic design and active space governance.

- **Avoid storing large files in the database.** Use external object storage for PDFs, images, or videos and store references (e.g., URLs) in the database.
- **Implement data retention policies.** Archive old transactional data to another schema, flat files, or cold storage if it's no longer queried frequently.

- **Partition large tables** using range or list partitioning to separate older data. Partitioning improves manageability and enables easier purging or archiving.
 - **Rotate logs regularly.** Besides binary logs, general logs and error logs should be rotated using tools like logrotate, especially in containerized environments.
 - **Monitor InnoDB transaction logs** (the ib_logfile* files). These are critical for crash recovery but should not grow indefinitely. If they become too large, you may need to reconfigure their size safely and restart the service.
 - **Store backups offsite.** Backups stored on the same volume as your live database can fill your disk. Use Elestio's backup tools to export backups to cloud storage or another disk.
-

Revision #2

Created 30 April 2025 09:02:37 by kaiwalya

Updated 30 April 2025 09:05:50 by kaiwalya