Checking Database Size and Related Issues

As your ClickHouse data grows especially with large analytical workloads or high-ingestion pipelines it’s important to track how storage is being used. Unchecked growth can lead to full disks, failed inserts, increased merge times, and slower queries. While Elestio handles the infrastructure, ClickHouse storage optimization and cleanup remain your responsibility. This guide explains how to inspect disk usage, analyze table size, detect inefficiencies, and manage ClickHouse storage effectively under a Docker Compose setup.

Checking Table Size and Disk Usage

ClickHouse stores data in columnar parts on disk, organized by partitions and merges. You can inspect disk consumption using SQL queries and Docker commands.

Check total disk space used by ClickHouse

From the host machine:

```
docker system df
```

Identify the Docker volume associated with ClickHouse, then check disk usage:

```
docker volume ls
sudo du -sh /var/lib/docker/volumes/<clickhouse_volume_name>/_data
```

Inspect space used per table

Connect to ClickHouse from the container:

```
docker-compose exec clickhouse clickhouse-client
```

Run:

```
SELECT
  database,
  table,
  formatReadableSize(sum(bytes_on_disk)) AS size_on_disk
FROM system.parts
```

```
WHERE active
GROUP BY database, table
ORDER BY sum(bytes_on_disk) DESC;
```

This shows total size used by each active table on disk.

View storage location inside container

ClickHouse typically writes data under /var/lib/clickhouse:

```
docker-compose exec clickhouse ls -lh /var/lib/clickhouse/store
```

This contains all table parts and metadata. Review sizes and delete orphaned data if needed.

Detecting Bloat and Inefficiencies

ClickHouse can accumulate unnecessary disk usage due to unoptimized merges, redundant partitions, or abandoned tables.

Check for unmerged parts

A high number of unmerged parts can slow down queries and increase disk usage:

```
SELECT
  database,
  table,
  count() AS part_count
FROM system.parts
WHERE active
GROUP BY database, table
ORDER BY part_count DESC;
```

Tables with many small parts may need a manual merge trigger.

Detect inactive or outdated parts

Look for inactive parts still occupying disk:

```
SELECT
  name,
  active,
  remove_time
FROM system.parts
```

```
WHERE active = 0
LIMIT 50;
```

These parts are safe to delete if they’re old and not part of ongoing operations.

Analyze storage by partition

To pinpoint heavy partitions:

```
SELECT
  table,
  partition_id,
  formatReadableSize(sum(bytes_on_disk)) AS size
FROM system.parts
WHERE active
GROUP BY table, partition_id
ORDER BY sum(bytes_on_disk) DESC;
```

Large partitions can indicate hot data or poor partitioning strategy.

Optimizing and Reclaiming ClickHouse Storage

ClickHouse provides several tools to optimize disk usage and clear unnecessary files.

Drop old partitions manually

For time-series or event tables, drop outdated partitions:

```
ALTER TABLE logs DROP PARTITION '2023-12';
```

Use partition pruning to maintain data freshness.

Optimize tables to force merges

To reduce part count and improve compression:

```
OPTIMIZE TABLE logs FINAL;
```

Use FINAL sparingly it can be resource-intensive.

Clean up old tables or unused databases

Drop stale or abandoned tables:

```
DROP TABLE old_analytics;
```

Drop entire databases if needed:

```
DROP DATABASE dev_test;
```

Always ensure no production data is affected.

Managing and Optimizing Files on Disk

ClickHouse stores metadata, parts, WAL logs, and temp files under /var/lib/clickhouse. You should monitor this path inside the container and from the host.

Monitor disk from inside container

```
docker-compose exec clickhouse du -sh /var/lib/clickhouse
```

To drill down:

```
docker-compose exec clickhouse du -sh /var/lib/clickhouse/*
```

Identify unexpectedly large directories like /store, /tmp, or /data.

Purge temporary files and logs

ClickHouse writes to /var/lib/clickhouse/tmp and /var/log/clickhouse-server/:

```
docker-compose exec clickhouse du -sh /var/lib/clickhouse/tmp
docker-compose exec clickhouse du -sh /var/log/clickhouse-server/
```

Clear if disk is nearing full. Rotate or truncate logs if necessary.

Clean WALs and outdated mutations

If mutations or insert queues are stuck:

```
SELECT * FROM system.mutations WHERE is_done = 0;
```

Investigate and resolve the root cause. Consider restarting ClickHouse after clearing safe logs.

Best Practices for ClickHouse Storage Management

- Use partitioning: Partition large tables by time (e.g., daily, monthly) to enable faster drops and better merge control.
- Archive old data: Move cold data to object storage (S3, etc.) or external databases for long-term storage.
- Avoid oversized inserts: Insert in smaller chunks to avoid bloating parts and reduce memory pressure during merges.
- Rotate logs: If ClickHouse logs to file, configure log rotation:

```
logging:
  driver: "json-file"
  options:
    max-size: "10m"
    max-file: "3"
```

- Use ZSTD compression: Prefer ZSTD over LZ4 for better compression ratio at the cost of slightly higher CPU.
- Monitor merges and disk pressure: Use system.metrics and system.events to track merge performance, part counts, and disk usage trends.
- Backup externally: Don’t store backups on the same disk. Use Elestio backup options to archive to remote or cloud storage.

Revision #2
Created 2025-06-11 11:55:20 UTC by kaiwalya
Updated 2025-06-11 12:00:48 UTC by kaiwalya