Checking Database Size and Related Issues

As your Redis data grows especially when using persistence modes like RDB or AOF it’s important to track how storage is being used. Unchecked growth can lead to full disks, failed writes, longer startup times, and backup complications. While Elestio handles the hosting, Redis storage tuning and cleanup remain your responsibility. This guide explains how to inspect keyspace size, analyze persistence files, detect unnecessary memory usage, and optimize Redis storage under a Docker Compose setup.

Checking Keyspace Usage and Persistence File Size

Redis doesn’t have schemas or tables, but its memory and disk footprint can be analyzed using built-in commands.

Check total memory used by Redis

From your terminal, connect to the container:

```
docker-compose exec redis redis-cli INFO memory
```

This displays current memory stats. Look for the used_memory_human and maxmemory fields to understand real usage versus limits.

Inspect key count and usage by database

```
docker-compose exec redis redis-cli INFO keyspace
```

Output looks like:

```
db0:keys=1250,expires=1200,avg_ttl=34560000
```

This tells you how many keys exist, how many have TTLs set, and their average lifespan. If most keys never expire, your dataset may grow indefinitely.

View on-disk file sizes

Inside the Redis container, persistent files live under /data:

```
docker-compose exec redis sh -c "ls -lh /data"
```

Check the sizes of:

- dump.rdb (if RDB is enabled)
- appendonly.aof (if AOF is enabled)

These files represent your on-disk dataset and can become large if not managed

Detecting Bloat and Unused Space

Redis may accumulate unnecessary memory usage due to expired keys not yet evicted, inefficient data structures, or infrequent AOF rewrites.

Estimate memory usage by key pattern

Redis doesn’t provide per-key memory stats natively, but you can sample keys and estimate memory usage:

```
docker-compose exec redis redis-cli --bigkeys
```

This scans a portion of the keyspace and reports the largest keys by type. If a single key is taking excessive space (e.g., a massive list or set), it may need to be split or purged.

Analyze memory per key (sample)

Use the MEMORY USAGE command to analyze specific keys:

```
docker-compose exec redis redis-cli MEMORY USAGE some:key
```

You can script this to scan high-traffic prefixes and locate heavy keys.

Check fragmentation

Redis may fragment memory, reducing efficiency:

```
docker-compose exec redis redis-cli INFO memory | grep fragmentation
```

A mem_fragmentation_ratio significantly above 1.2 suggests internal fragmentation.

Optimizing and Reclaiming Redis Storage

Once you've identified memory-heavy keys or large persistence files, Redis offers several tools to optimize space usage.

Trigger AOF rewrite (compacts the appendonly file)

If AOF is enabled, it grows over time. To reduce its size:

```
docker-compose exec redis redis-cli BGREWRITEAOF
```

This background process creates a smaller version of the AOF file without data loss.

Delete or expire unused keys

Manually delete stale keys or add TTLs to ensure automatic cleanup:

```
docker-compose exec redis redis-cli DEL obsolete:key
```

Or set expiration:

```
docker-compose exec redis redis-cli EXPIRE session:1234 3600
```

Use patterns to delete multiple keys (carefully!):

```
docker-compose exec redis redis-cli --scan --pattern "temp:*" | xargs -n 100 redis-cli DEL
```

> Avoid FLUSHALL or bulk deletes in production unless absolutely necessary.

Tune maxmemory and eviction policy

To enforce automatic eviction when nearing memory limits, In redis.conf (mounted via Docker volume):

```
maxmemory 512mb
maxmemory-policy allkeys-lru
```

Restart the container to apply changes. This keeps Redis performant under constrained storage.

Managing and Optimizing Redis Files on Disk

Monitor data directory inside Docker

Redis typically writes to /data in the container (mapped from a host volume). Check usage from the host:

```
docker system df
```

List all Docker volumes:

```
docker volume ls
```

Check Redis volume size (replace <volume_name>):

```
sudo du -sh /var/lib/docker/volumes/<volume_name>/_data
```

Clean up RDB snapshots and old backups

RDB snapshots (e.g. dump.rdb) are stored in /data. Clean up old or unneeded ones manually:

```
docker-compose exec redis rm /data/dump-<timestamp>.rdb
```

Ensure backups are offloaded to external storage and not stored alongside the live database.

Best Practices for Redis Storage Management

- **Use TTLs liberally**: Set expiration on all temporary/session keys to prevent unbounded growth.
- **Avoid storing large binary blobs**: Store images, files, or videos outside Redis. Use Redis for metadata only.
- **Rotate logs**: If Redis logs to file (e.g., /var/log/redis.log), rotate them via Docker logging options or tools like logrotate.
  In docker-compose.yml

```
logging:
  driver: "json-file"
```

```
  options:
    max-size: "10m"
    max-file: "3"
```

- **Use efficient data structures**: Prefer HASH or SET over storing large JSON blobs as strings.
- **Monitor AOF size and compaction frequency**: If AOF is growing too fast, adjust these in redis.conf:

```
auto-aof-rewrite-percentage 100
auto-aof-rewrite-min-size 64mb
```

- **Archive analytics data**: For time-series or metrics data, periodically move old entries to cold storage.
- **Back up to offsite storage**: Avoid keeping snapshots on the same disk or volume. Use Elestio’s backup integrations to store them in cloud or remote storage.

Revision #1
Created 2025-05-20 11:38:57 UTC
Updated 2025-05-20 12:08:28 UTC