Checking Database Size and Related Issues

As your Valkey data grows especially when using persistence modes like RDB or AOF it’s essential to monitor how disk and memory resources are consumed. Uncontrolled growth can result in full disks, write failures, longer restarts, and issues with snapshot backups. While Elestio handles infrastructure hosting, managing storage cleanup and optimization is the user’s responsibility. This guide shows how to inspect keyspace usage, analyze persistence files, detect memory bloat, and tune your Valkey deployment under Docker Compose.

Checking Keyspace Usage and Persistence File Size

Like Redis, Valkey doesn’t have schemas or tables but provides insights through built-in commands and memory metrics.

Check total memory used by Valkey

Connect to the container:

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

Look at used_memory_human and maxmemory to understand current usage and configured limits.

Inspect key count and TTL stats

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

You’ll see entries like:

```
db0:keys=2400,expires=2100,avg_ttl=36000000
```

This helps identify how many keys are temporary and whether the dataset will grow indefinitely.

View on-disk file sizes

Valkey writes persistence files to /data inside the container:

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

Check the size of dump.rdb, appendonly.aof, and any temporary files.

Detecting Bloat and Unused Space

Valkey supports Redis commands and adds community-focused improvements, but it can still suffer from memory inefficiencies if not monitored properly.

Estimate memory usage by key pattern

```
docker-compose exec valkey valkey-cli --bigkeys
```

This reveals large keys by data type, helping you spot high-memory structures like oversized lists or sets.

Analyze memory per key (manual sample)

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

This helps profile storage-heavy keys by prefix or type.

Check memory fragmentation

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

If mem_fragmentation_ratio is over 1.2, it may indicate inefficient memory allocation.

Optimizing and Reclaiming Valkey Storage

Once you've identified bloated memory areas or oversized persistence files, you can apply optimizations.

Compact the AOF file

```
docker-compose exec valkey valkey-cli BGREWRITEAOF
```

This rewrites and reduces the size of appendonly.aof.

Delete unused keys or apply TTLs

```
docker-compose exec valkey valkey-cli DEL obsolete:key
docker-compose exec valkey valkey-cli EXPIRE session:1234 3600
```

To bulk-delete keys by pattern (use with caution):

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

Configure eviction policies

In your mounted valkey.conf:

```
maxmemory 1gb
maxmemory-policy allkeys-lru
```

Restart the container to apply these changes. This ensures automatic cleanup when memory thresholds are exceeded.

Best Practices for Valkey Storage Management

- Use TTLs on non-permanent keys: Set expiration on cache/session data to avoid unbounded key growth.
- Avoid storing binary files in Valkey: Keep large files (images, documents, etc.) in object storage. Use Valkey only for metadata.
- Rotate container logs: In your docker-compose.yml:

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

- Use compact data structures: Favor HASH, SET, or ZSET over storing entire JSON blobs as STRING.
- Monitor and control AOF size: Configure AOF rewrite frequency in valkey.conf:

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

- Archive old analytical data: Periodically move old metrics, logs, or time-series entries to cold storage.
- Externalize backups: Use Elestio’s backup features or configure external volumes/cloud storage to avoid accumulating snapshots on the same disk used for live data.

Revision #2
Created 2025-07-04 10:17:49 UTC
Updated 2025-07-04 10:38:24 UTC