How-To Guides

- Creating a Database
- Upgrading to Major Version
- Installing and Updating an Extension
- Creating Manual Backups
- Restoring a Backup
- Identifying Slow Queries
- Detect and terminate long-running queries
- Preventing Full Disk Issues
- Checking Database Size and Related Issues

Creating a Database

Redis is a popular in-memory key-value data store known for its speed, flexibility, and support for a wide range of data structures. Setting up Redis properly is essential to ensure high availability, data persistence, and performance in modern applications. This guide walks through different ways to run and connect to Redis: using the Redis CLI, using Docker containers, and using Redis CLI tools. It also emphasizes best practices that should be followed at each step.

Creating Using redis-cli

The Redis command-line interface (redis-cli) is a built-in tool that allows direct interaction with a Redis server. It supports connecting to local or remote Redis instances and executing all supported Redis commands interactively or non-interactively.

Connect to Redis:

If you're running Redis locally (e.g., from a system package or Docker container on your machine), you can simply run the CLI tool with no arguments:

```
redis-cli
```

For remote connections, you need to provide the hostname or IP address, the port number (default is 6379), and the password if the instance is protected:

```
redis-cli -h <host> -p <port> -a <password>
```

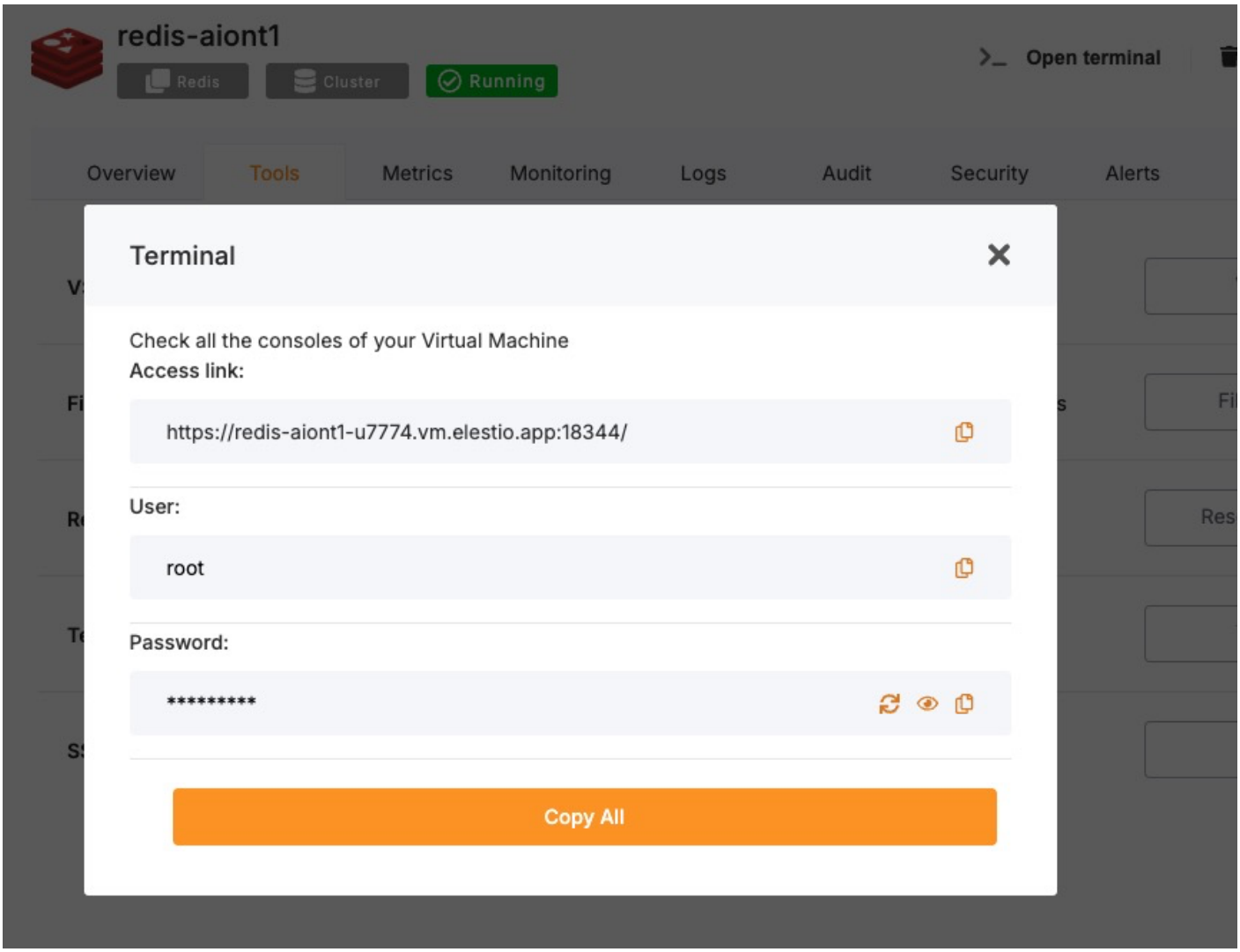
You'll be dropped into the Redis shell, where commands can be executed directly.

Running Redis Using Docker

Docker is a widely-used tool for running applications in isolated environments called containers. Redis can be deployed in a container for fast and consistent setup across different environments, making it ideal for both development and production.

Access Elestio Terminal

If you're using Elestio for your Redis deployment, log into the Elestio dashboard. Go to your Redis service, then navigate to **Tools > Terminal**. This opens a browser-based terminal already configured for the correct environment.



Once in the terminal, change the directory to the project root where the Docker services are defined. This is typically:

```
cd /opt/app/
```

Access the Redis Container Shell

Elestio services are managed using Docker Compose, which orchestrates multiple containers. To enter the Redis container's shell and run Redis commands, use the following:

```
docker-compose exec redis bash
```

This command opens a shell inside the running Redis container.

Access Redis CLI from Within the Container

Inside the Redis container, the redis-cli command is already available. Use it to access the Redis instance. If authentication is enabled, supply the password using the -a flag:

```
redis-cli -a <password>
```

You will be connected directly to the Redis server running inside the container.

Test Connectivity

Use a simple SET command to store a value and then retrieve it using GET to ensure Redis is functioning properly:

```
set testkey "Hello Redis"
get testkey
```

Expected output:

```
"Hello Redis"
```

This verifies read/write operations are working as expected.

Connecting Using redis-cli in Scripts

redis-cli isn’t just for interactive use — it can also be used within shell scripts and automated pipelines. This is useful for deployment tasks, monitoring scripts, and health checks.

For example, to set a key from a script or cron job:

```
redis-cli -h <host> -p <port> -a <password> SET example_key "example_value"
```

This command connects to the Redis server and sets the specified key in a single line, suitable for automation.

Best Practices for Setting Up Redis

Use Meaningful Key Naming Conventions

Organizing your Redis data is essential for maintainability and clarity. Use descriptive keys and adopt a namespace-style format using colons (:) to group related keys:

```
user:1001:profile
order:2023:total
```

This makes it easier to debug, analyze, and migrate data in the future.

Follow Consistent Data Structures

Redis supports multiple data types: strings, hashes, lists, sets, and sorted sets. Choose the most appropriate structure for your use case, and apply it consistently. For example, use hashes to store user attributes and lists for ordered items.

Inconsistent or incorrect use of data structures can lead to performance issues and logic errors.

Enable Authentication and TLS

Security is critical in production environments. Always set a strong password using the requirepass directive in redis.conf, and consider enabling TLS for encrypted communication if Redis is exposed over a network.

Example `redis.conf` settings:

```
requirepass strong_secure_password
tls-port 6379
tls-cert-file /path/to/cert.pem
tls-key-file /path/to/key.pem
```

This helps prevent unauthorized access and secures data in transit.

Configure Persistence Options

Redis is in-memory, but it supports two main persistence options:

- **RDB (snapshotting)**: takes periodic snapshots of the data.
- **AOF (Append Only File)**: logs each write operation for more durable persistence.

Set these options in redis.conf:

```
save 900 1
appendonly yes
appendfsync everysec
```

Use AOF for durability and RDB for faster restarts — or combine both for a balance.

Monitor and Tune Performance

Use Redis's built-in tools (INFO, MONITOR, SLOWLOG) to analyze behavior, identify slow queries, and monitor memory usage. This helps maintain performance and plan for scaling.

External tools like **RedisInsight**, **Prometheus**, or **Grafana** can visualize metrics and alert on anomalies.

Common Issues and Their Solutions

Issue	Cause	Solution
NOAUTH Authentication required.	Connecting to a password-protected Redis without a password	Use the -a <password> flag or the AUTH command before other commands
ERR Client sent AUTH, but no password is set	The Redis server does not require a password	Remove the -a flag or check requirepass in redis.conf
Can’t connect to Redis on ‘localhost’	Redis is not running or using the wrong port	Start Redis and check redis.conf or Docker port mappings
Docker Redis container refuses connections	Container is not ready or misconfigured network	Check logs using docker-compose logs redis and verify port exposure
Data not persisted after restart	Persistence settings are disabled	Enable AOF or RDB in redis.conf

Upgrading to Major Version

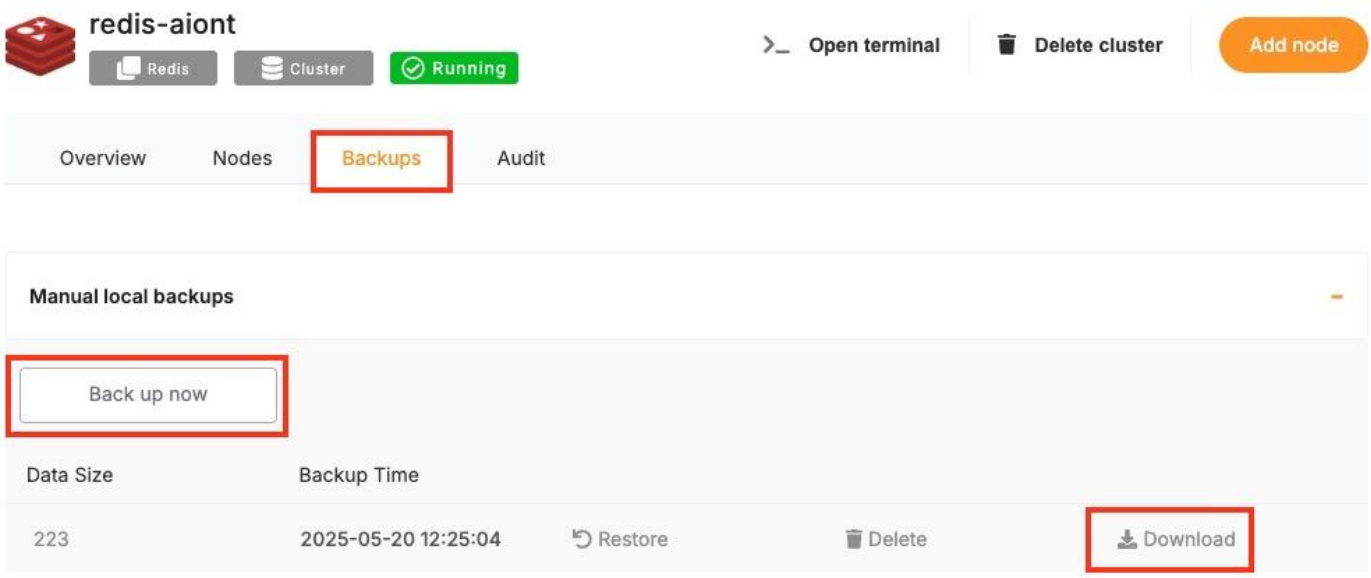
Upgrading a database service on Elestio can be done without creating a new instance or performing a full manual migration. Elestio provides a built-in option to change the database version directly from the dashboard. This is useful for cases where the upgrade does not involve breaking changes or when minimal manual involvement is preferred. The version upgrade process is handled by Elestio internally, including restarting the database service if required. This method reduces the number of steps involved and provides a way to keep services up to date with minimal configuration changes.

Log In and Locate Your Service

To begin the upgrade process, log in to your Elestio dashboard and navigate to the specific database service you want to upgrade. It is important to verify that the correct instance is selected, especially in environments where multiple databases are used for different purposes such as staging, testing, or production. The dashboard interface provides detailed information for each service, including version details, usage metrics, and current configuration. Ensure that you have access rights to perform upgrades on the selected service. Identifying the right instance helps avoid accidental changes to unrelated environments.

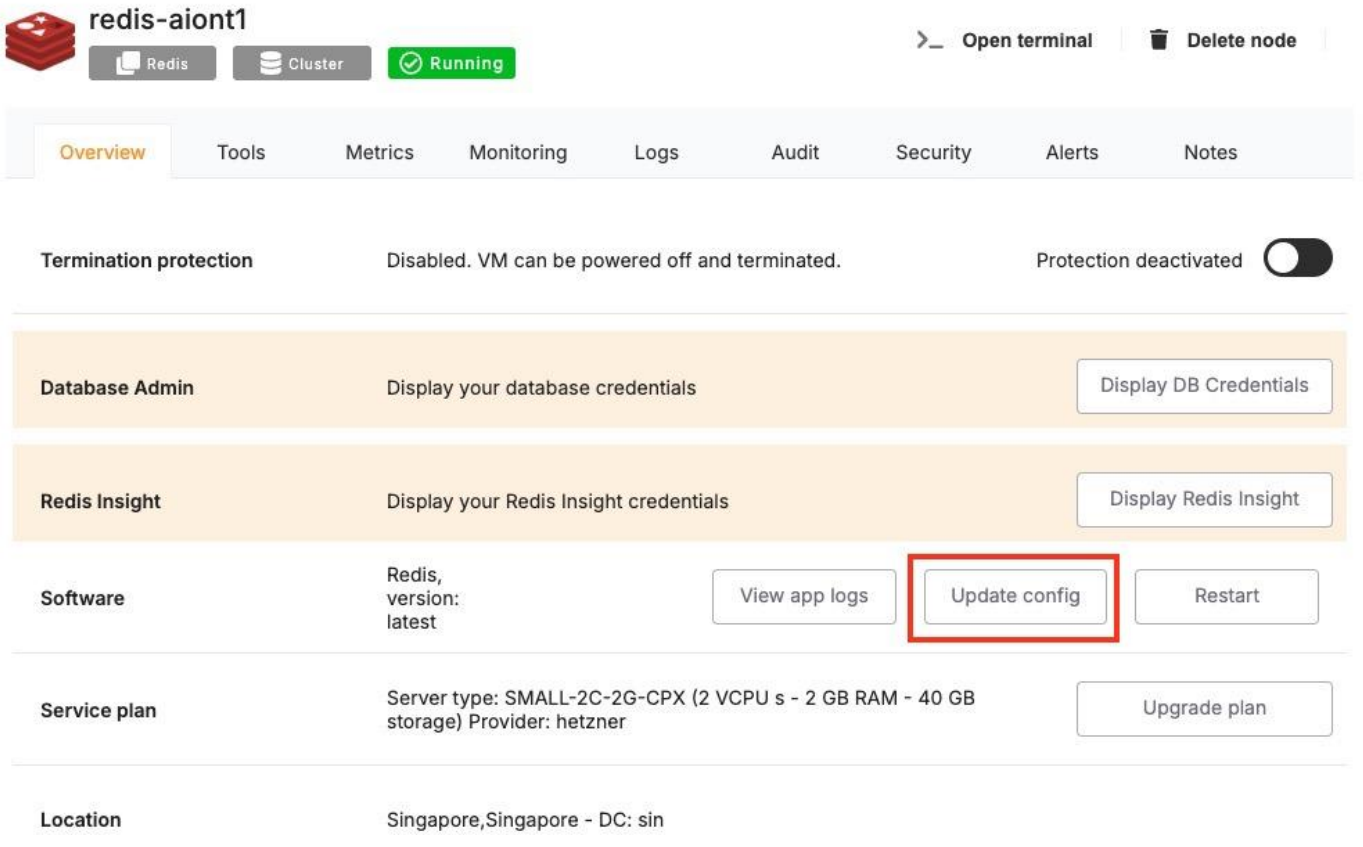
Back Up Your Data

Before starting the upgrade, create a backup of your database. A backup stores the current state of your data, schema, indexes, and configuration, which can be restored if something goes wrong during the upgrade. In Elestio, this can be done through the **Backups** tab by selecting **Back up now** under Manual local backups and **Download** the backup file. Scheduled backups may also be used, but it is recommended to create a manual one just before the upgrade. Keeping a recent backup allows quick recovery in case of errors or rollback needs. This is especially important in production environments where data consistency is critical.

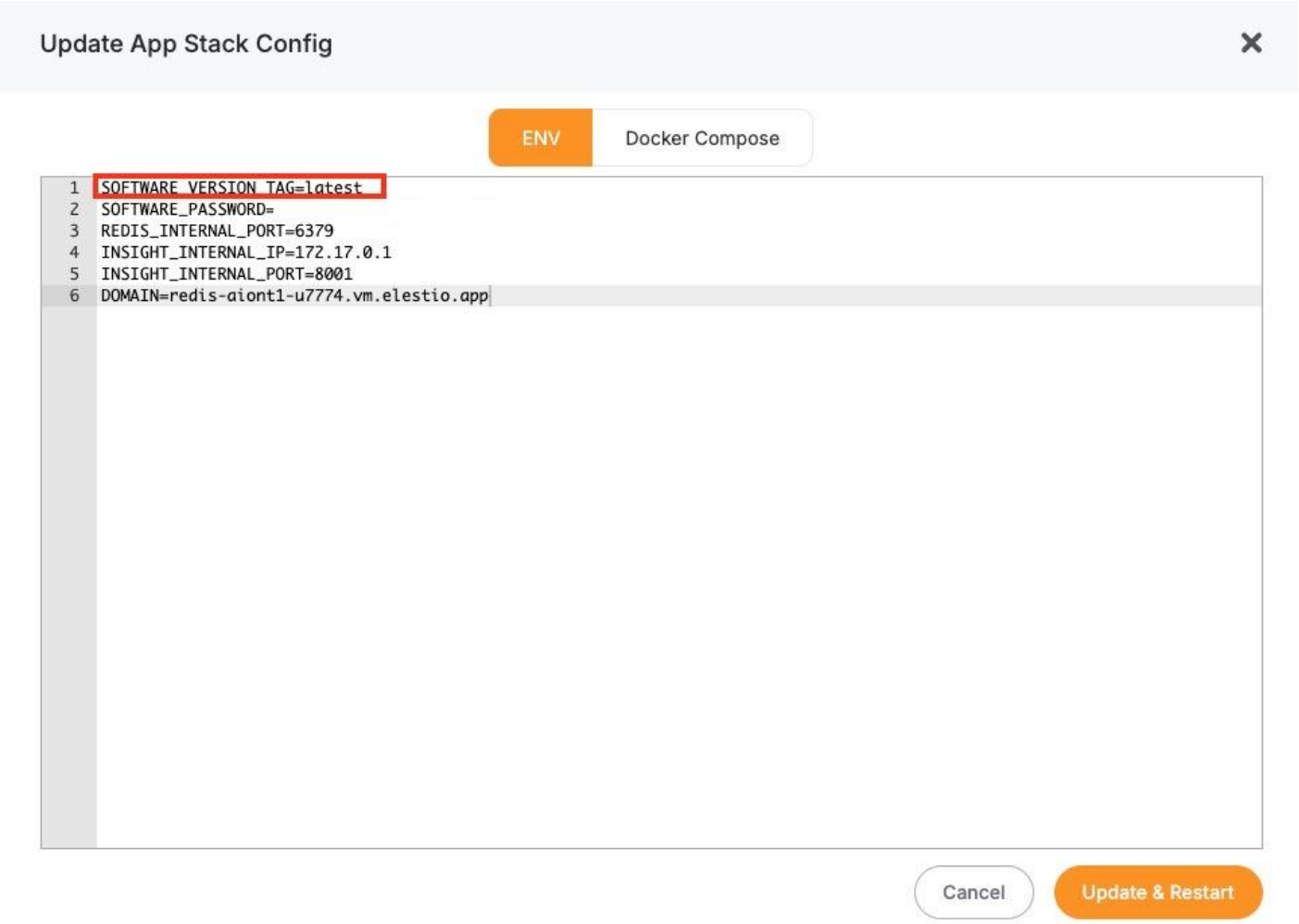


Select the New Version

Once your backup is secure, proceed to the **Overview** and then **Software > Update config** tab within your database service page.



Here, you'll find an option labeled **ENV**. In the **ENV** menu, change the desired database version to `SOFTWARE_VERSION`. After confirming the version, Elestio will begin the upgrade process automatically. During this time, the platform takes care of the version change and restarts the database if needed. No manual commands are required, and the system handles most of the operational aspects in the background.



Monitor the Upgrade Process

The upgrade process may include a short downtime while the database restarts. Once it is completed, it is important to verify that the upgrade was successful and the service is operating as expected. Start by checking the logs available in the Elestio dashboard for any warnings or errors during the process. Then, review performance metrics to ensure the database is running normally and responding to queries. Finally, test the connection from your client applications to confirm that they can interact with the upgraded database without issues.

Installing and Updating an Extension

Redis supports **modules** to extend core functionality with new data types, commands, or algorithms. These modules behave like plugins in other systems and are loaded at server startup. Examples include RedisBloom, RedisTimeSeries, RedisJSON, and RedisSearch.

In Elestio-hosted Redis instances or any Docker Compose-based setup, modules can be loaded by specifying them in the service configuration. This guide walks through how to install, load, and manage Redis modules using Docker Compose, along with common issues and best practices.

Installing and Enabling Redis Modules

Redis modules are typically compiled as shared object (.so) files and must be loaded at server startup using the `--loadmodule` option. These module files are mounted into the container and referenced from within the container's file system. To use a module like RedisBloom in a Docker Compose setup:

Update docker-compose.yml

Mount the module file into the container and load it:

```
services:
  redis:
    image: redis/redis-stack-server:latest
    volumes:
      - ./modules/redisbloom.so:/data/redisbloom.so
    command: ["redis-server", "--loadmodule", "/data/redisbloom.so"]
    ports:
      - "6379:6379"
```

Here:

- `./modules/redisbloom.so` is the local path on your host machine.
- `/data/redisbloom.so` is the path inside the container.

Make sure the .so file exists in the specified directory before running Docker Compose.

Restart the Redis Service

After updating the Compose file, restart the service:

```
docker-compose down
docker-compose up -d
```

This will reload Redis with the specified module.

Verify the Module is Loaded

Once Redis is running, connect to it using redis-cli:

```
docker-compose exec redis redis-cli -a <yourPassword>
```

Run the following command:

```
MODULE LIST
```

Expected output:

```
1) 1) "name"
   2) "bf"
   3) "ver"
   4) (integer) 20207
```

This confirms the module (in this case, bf for RedisBloom) is loaded and active.

Checking Module Availability & Compatibility

Redis modules must match the Redis server version and platform. You can verify compatibility through the module's documentation or by testing it in a local development setup before using it in production.

To inspect module-related details:

```
INFO MODULES
```

To verify the correct Redis image is being used:

```
docker-compose exec redis redis-server --version
```

If a module fails to load, check the container logs:

```
docker-compose logs redis
```

This often reveals missing paths or compatibility issues.

Updating or Unloading Modules

Unlike MySQL, Redis does **not** support dynamic unloading of modules once loaded. To update or remove a module:

1. **Stop the container:**

```
docker-compose down
```

2. **Edit docker-compose.yml:**

   - Change the .so file path if updating the module.
   - Remove the `--loadmodule` line if unloading the module.

3. **Restart the container:**

```
docker-compose up -d
```

Always test updated modules in staging before applying to production.

Troubleshooting Common Module Issues

Issue	Cause	Resolution
Redis fails to start	Incorrect module path or incompatible binary	Check docker-compose logs redis and verify the .so path and architecture
MODULE command not recognized	Using a Redis image without module support	Use an image like redis/redis-stack-server which supports modules

Issue	Cause	Resolution
“Can’t open .so file”	Volume not mounted or permission denied	Ensure the .so file exists locally and is readable by Docker
Module not appearing in MODULE LIST	Module failed to load silently	Double-check command and container logs
Commands from the module not recognized	Module not loaded properly or incompatible	Validate Redis version and module compatibility

Security Considerations

Redis modules execute native code with the same privileges as Redis itself. Only load trusted, vetted modules from official sources. Avoid uploading or executing arbitrary .so files from unknown authors. In multi-tenant or exposed environments, module misuse could lead to instability or security risks. Ensure the redis user inside the container has limited privileges, and module directories have appropriate permissions.

Creating Manual Backups

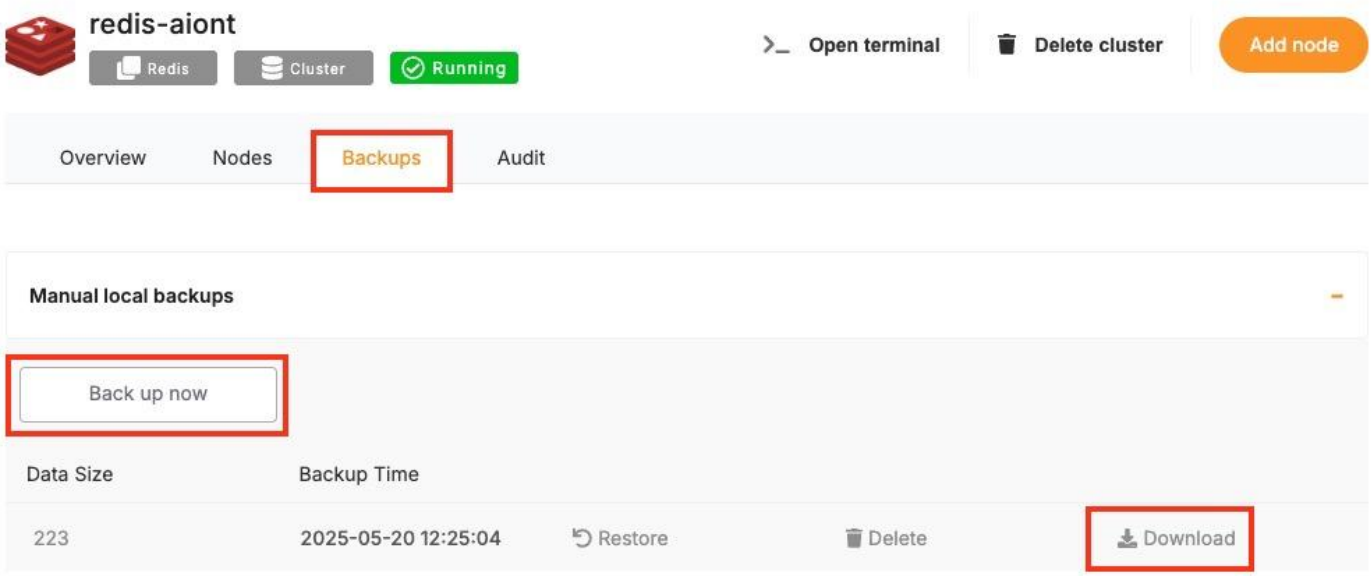
Regular backups are essential when running a Redis deployment especially if you're using it for persistent data. While Elestio handles automated backups by default, you may want to create manual backups before configuration changes, retain a local archive, or test backup automation. This guide walks through multiple methods for creating Redis backups on Elestio, including dashboard snapshots, command-line approaches, and Docker Compose-based setups. It also explains backup storage, retention, and automation using scheduled jobs.

Manual Service Backups on Elestio

If you're using Elestio's managed Redis service, the simplest way to perform a full backup is directly through the Elestio dashboard. This creates a snapshot of your current Redis dataset and stores it in Elestio's infrastructure. These snapshots can be restored later from the same interface, which is helpful when making critical changes or testing recovery workflows.

To trigger a manual Redis backup on Elestio:

1. Log in to the Elestio dashboard.
2. Navigate to your Redis service or cluster.
3. Click the **Backups** tab in the service menu.
4. Choose **Back up now** to generate a manual snapshot.



This method is recommended for quick, reliable backups without needing to use the command line.

Manual Backups Using Docker Compose

If your Redis instance is deployed via Docker Compose (as is common on Elestio-hosted environments), you can manually back up Redis by copying its internal snapshot files. Redis persistence is managed through **RDB (Redis Database)** and optionally **AOF (Append-Only File)** logs, both of which reside in the container's filesystem.

Access Elestio Terminal

Go to your deployed Redis service in the Elestio dashboard, navigate to **Tools > Terminal**, and log in using the credentials provided.

Locate the Redis Container Directory

Navigate to your app directory:

```
cd /opt/app/
```

This is the working directory of your Docker Compose project, which contains the docker-compose.yml file.

Trigger an RDB Snapshot (Optional)

Redis typically saves snapshots automatically based on configuration, but you can force one manually:

```
docker-compose exec redis redis-cli SAVE
```

This command triggers an immediate snapshot. The resulting file is usually called dump.rdb.

Copy Backup Files from the Container

Use docker cp to copy the snapshot file (and optionally the AOF file, if enabled) from the container to your host system:

```
docker cp $(docker-compose ps -q redis):/data/dump.rdb ./backup_$(date +%F).rdb
```

If AOF persistence is also enabled (via appendonly yes in redis.conf), back up the AOF log as well:

```
docker cp $(docker-compose ps -q redis):/data/appendonly.aof ./appendonly_$(date +%F).aof
```

This gives you complete Redis data snapshots for storage or future recovery.

Backup Storage & Retention Best Practices

After creating backups, it’s important to store them securely and manage retention properly. Redis backups are binary files and can be quite compact (RDB) or larger and more frequent (AOF), depending on configuration.

Guidelines to Follow:

- Use clear naming: `redis_backup_2025_05_19.rdb`
- Store off-site or on cloud storage (e.g. AWS S3, Backblaze, encrypted storage).
- Retain: 7 daily backups, 4 weekly backups, and 3–6 monthly backups.
- Automate old file cleanup with cron jobs or retention scripts.
- Optionally compress backups with gzip or xz to reduce space.

Automating Redis Backups (cron)

Manual backup commands can be scheduled using tools like cron on Linux-based systems. This allows you to regularly back up your database without needing to run commands manually. Automating the process also reduces the risk of forgetting backups and ensures more consistent retention.

Example: Daily Backup at 3 AM

1. Edit the crontab:

```
crontab -e
```

2. Add a job like:

```
0 3 * * * docker-compose -f /opt/app/docker-compose.yml exec redis redis-cli SAVE && \
docker cp $(docker-compose -f /opt/app/docker-compose.yml ps -q redis):/data/dump.rdb
/backups/redis_backup_$(date +\%F).rdb
```

Make sure /backups/ exists and is writable by the cron user.

You can also compress the file or upload to cloud storage in the same script:

```
gzip /backups/redis_backup_$(date +\%F).rdb
rclone copy /backups/remote-dir/ remote:redis-backups
```

Backup Format and Restore Notes

Format	Description	Restore Method
`dump.rdb`	Binary snapshot of full dataset	Stop Redis, replace dump.rdb, then restart Redis
`appendonly.aof`	Append-only command log	Stop Redis, replace AOF file, then restart Redis

To restore from a backup:

1. Stop Redis (docker-compose down)
2. Replace the corresponding file in the volumes or /data directory.
3. Restart Redis (docker-compose up -d)

Restoring a Backup

Restoring Redis backups is essential for disaster recovery, staging environment duplication, or rolling back to a known state. Elestio supports backup restoration both through its web dashboard and manually through Docker Compose and command-line methods. This guide explains how to restore Redis backups from RDB or AOF files, covering both full and partial restore scenarios, and includes solutions for common restoration issues.

Restoring from a Backup via Terminal

This method applies when you have an RDB (dump.rdb) or AOF (appendonly.aof) file from a previous backup. To restore the backup, you replace the existing Redis data file(s) inside the data directory used by the container. Redis loads this data at startup, making it essential to stop the server before replacing the files.

Stop the Redis Container

Shut down the Redis container cleanly to avoid file corruption:

```
docker-compose down
```

Replace the Backup File

Move your backup file into the appropriate location inside the Redis volume. Assuming you have a backup named backup_2025_05_19.rdb:

```
cp ./backup_2025_05_19.rdb /opt/app/data/dump.rdb
```

Make sure this file path corresponds to the volume used in your docker-compose.yml. For example:

```
volumes:
  - ./data:/data
```

If you’re restoring an AOF file, replace appendonly.aof instead:

```
cp ./appendonly_2025_05_19.aof /opt/app/data/appendonly.aof
```

Restart Redis

Start Redis again so it loads the restored data file:

```
docker-compose up -d
```

Redis will automatically load dump.rdb or appendonly.aof depending on your configuration (set in redis.conf with appendonly yes/no).

Restoring via Docker Compose Terminal

If you prefer working inside the container, you can also copy the file directly into the Redis container using Docker commands.

Copy the Backup File into the Container

```
docker cp ./backup_2025_05_19.rdb $(docker-compose ps -q redis):/data/dump.rdb
```

If restoring an AOF file:

```
docker cp ./appendonly_2025_05_19.aof $(docker-compose ps -q redis):/data/appendonly.aof
```

Restart Redis Inside Docker Compose

```
docker-compose restart redis
```

Redis will detect the updated data file and load it during startup.

Partial Restores in Redis

Redis does not natively support partial restores like MySQL. However, you can achieve similar outcomes with the following strategies:

Restore Selected Keys via Redis CLI

If you exported individual key-value pairs using the redis-cli --rdb or similar logic, you can use a script to reinsert only those keys.

Example using redis-cli and a JSON/CSV conversion:

```
cat keys_to_restore.txt | while read key; do
  value=$(cat dump.json | jq -r ".\"$key\"")
  redis-cli SET "$key" "$value"
done
```

This approach assumes you have extracted individual key-value pairs into a format suitable for scripting.

Restore from A Partial AOF

If your append-only file includes only a subset of commands, Redis will replay those on startup. You can prepare a stripped-down AOF file for specific keys or operations, then follow the full AOF restore method described above.

Common Errors & How to Fix Them

Restoring Redis data can fail for a few specific reasons, especially related to permissions, missing config values, or service conflicts. Here are some frequent issues and how to solve them.

1. NOAUTH Authentication Required

```
(error) NOAUTH Authentication required.
```

Cause: You're attempting to issue commands or restore data into a Redis instance that requires authentication.

Resolution: Always provide the password with your commands:

```
redis-cli -a yourpassword
```

For automated scripts, use:

```
redis-cli -a "$REDIS_PASSWORD" < restore_script.txt
```

2. Redis Fails to Start After Restore

```
Fatal error loading the DB: Invalid RDB format
```

Cause: Corrupted or incompatible `dump.rdb` or `appendonly.aof` file.

Resolution: Ensure the backup file matches the Redis version you're using. Try restoring with a version of Redis that matches the backup environment.

3. Data Not Restored

Cause: Redis is configured to use AOF, but only an RDB file was restored or vice versa.

Resolution: Confirm your redis.conf or container command: entry defines which persistence method is enabled:

```
appendonly yes  # For AOF
```

```
appendonly no   # For RDB
```

Make sure the correct file (either dump.rdb or appendonly.aof) is in /data.

4. Permission Denied When Copying Files

```
cp: cannot create regular file ‘/opt/app/data/dump.rdb’: Permission denied
```

Resolution: Ensure your terminal session or script has write access to the target directory. Use sudo if needed:

```
sudo cp ./backup.rdb /opt/app/data/dump.rdb
```

Identifying Slow Queries

Slow commands can impact Redis performance, especially under high load or when poorly optimized operations are used. Whether you’re using Redis on Elestio through the dashboard, accessing it inside a Docker Compose container, or connecting via CLI tools, Redis provides native tooling to monitor and troubleshoot performance issues. This guide explains how to capture slow operations using the Redis slow log, analyze command latency, and optimize performance through configuration and query changes.

Inspecting Slow Commands from the Terminal

Redis includes a built-in **slowlog** feature that tracks commands exceeding a configured execution time threshold. This is useful for identifying operations that may block the server or cause application latency.

Connect to your Redis instance via terminal

Use the Redis CLI to connect to your instance:

```
redis-cli -h <host> -p <port> -a <password>
```

> “ Replace <host>, <port>, and <password> with your Redis credentials from the Elestio dashboard.

View the slowlog threshold

Check the threshold that defines a “slow” command (in microseconds):

```
CONFIG GET slowlog-log-slower-than
```

The default is 10000 (10 milliseconds). Any command exceeding this will be logged.

View the slow query log

To inspect recent slow commands:

```
SLOWLOG GET 10
```

This shows the 10 most recent slow commands. Each entry includes the execution time, timestamp, and command details.

Analyzing Inside Docker Compose

If your Redis instance is deployed with Docker Compose, slow command inspection can be done inside the running container environment.

Access the Redis container

Open a shell inside the container:

```
docker-compose exec redis bash
```

Then connect to Redis using:

```
redis-cli -a $REDIS_PASSWORD
```

Make sure the REDIS_PASSWORD environment variable is defined in your Docker Compose file.

Check and adjust the slowlog threshold

You can view or change the slowlog threshold dynamically:

```
CONFIG SET slowlog-log-slower-than 10000
```

Set a lower threshold (e.g., 5000) temporarily to capture more entries during testing.

Check how many entries are stored

The number of slowlog entries stored is configurable:

```
CONFIG GET slowlog-max-len
```

To increase the history size:

```
CONFIG SET slowlog-max-len 256
```

This allows storing more slow command logs for better visibility.

Using the Latency Monitoring Feature

Redis also includes latency monitoring tools that track spikes and identify root causes.

Enable latency monitoring

Latency tracking is often enabled by default. You can manually inspect events with:

```
LATENCY DOCTOR
```

This command gives a report of latency spikes and their possible causes (e.g., slow commands, forks, or blocked I/O).

View latency history for specific events

To inspect latency for a specific category:

```
LATENCY HISTORY command
```

Common tracked events include command, fork, aof-write, etc.

Understanding and Resolving Common Bottlenecks

Redis performance can degrade due to specific patterns of usage, large keys, blocking commands, or non-optimized pipelines.

Common causes of slow commands:

- **Large key operations**: Commands like LRANGE, SMEMBERS, HGETALL on large datasets.
- **Blocking operations**: Commands like BLPOP, BRPOP, or Lua scripts with long loops.
- **Forking overhead**: Caused by background saves or AOF rewrites.

Best practices to avoid slow commands:

- Use **SCAN** instead of **KEYS** for iteration.
- Limit result sizes from large structures (e.g., use LRANGE 0 99 instead of full LRANGE).
- Use **pipelining** to batch requests and reduce round trips.
- Avoid **multi-key** operations when possible in a clustered setup.

Optimizing with Configuration Changes

Performance tuning can also involve modifying Redis settings related to memory, persistence, and networking.

Update these settings via redis.conf or dynamically with CONFIG SET:

```
CONFIG SET maxmemory-policy allkeys-lru
CONFIG SET save ""
```

Use caution with persistence settings. Disabling RDB or AOF improves performance but removes durability.

Detect and terminate long-running queries

Long-running commands in Redis can block the single-threaded event loop, causing delayed responses or complete unresponsiveness in production environments like Elestio. Monitoring and handling these commands is critical for maintaining performance and reliability. This guide explains how to detect, analyze, and terminate blocking or slow commands in Redis using terminal tools, Docker Compose setups, and Redis's built-in logging features. It also includes prevention strategies to avoid performance bottlenecks in the future.

Monitoring Long-Running Commands

Redis does not support multitasking like traditional SQL databases, so any command that takes too long blocks the entire server. To inspect active commands and see which clients may be running long operations, use the Redis CLI.

Check active clients and their current commands

```
redis-cli -h <host> -p <port> -a <password> CLIENT LIST
```

This command shows all connected clients, including their IP address, command in progress (cmd), idle time, and total duration. Focus on clients with high idle or age values while still actively running commands.

Detect current command load using MONITOR

To observe commands in real time:

```
redis-cli -a <password> MONITOR
```

This outputs every operation in real time. It's useful for spotting blocking commands but should be used only in staging or during short troubleshooting sessions, as it consumes significant CPU.

Terminating Problematic Commands Safely

Redis provides tools to close problematic connections or interrupt Lua scripts that run for too long.

Kill a specific client connection

If a client is running a blocking or long operation, you can terminate its connection using its client ID:

```
CLIENT KILL ID <id>
```

> You can find the <id> from the CLIENT LIST command.

This will drop the connection and stop any running command associated with that client.

Stop a long-running Lua script

If a Lua script is stuck or taking too long:

```
SCRIPT KILL
```

This stops the currently executing script. If the script has modified data, Redis will return an error to avoid leaving the database in an inconsistent state.

> If the script is not killable (e.g., during a write operation), Redis will return an error. Always use SCRIPT KILL cautiously.

Managing Inside Docker Compose

If your Redis service is running inside a Docker Compose setup on Elestio, you'll need to access the container before you can inspect or kill commands.

Access the Redis container

```
docker-compose exec redis bash
```

Inside the container, connect to Redis using:

```
redis-cli -a $REDIS_PASSWORD
```

Then, use CLIENT LIST, SCRIPT KILL, or CLIENT KILL just like from the host.

Using the Redis Slowlog Feature

Redis includes a built-in slowlog that logs commands that exceed a specific execution threshold.

Enable and configure slowlog in redis.conf

```
slowlog-log-slower-than 10000        # Log commands slower than 10ms
slowlog-max-len 128                  # Keep 128 slow entries
```

Update these settings in redis.conf, or set them at runtime:

```
CONFIG SET slowlog-log-slower-than 10000
CONFIG SET slowlog-max-len 128
```

View the slowlog

```
SLOWLOG GET 10
```

This shows the 10 most recent slow commands with their timestamp, execution time, and command details.

Clear the slowlog

```
SLOWLOG RESET
```

Use this to reset the log after reviewing or during maintenance.

Analyzing Command Latency Over Time

Redis includes latency tracking features to help you understand when and why delays occur.

Generate a diagnostic latency report

```
LATENCY DOCTOR
```

This gives you a summary of observed latency spikes and their causes (e.g., command execution, AOF rewrite, background saves).

View detailed latency history by event

```
LATENCY HISTORY command
```

You can replace command with any tracked event like fork, aof-write, or expire-cycle.

Best Practices to Prevent Long-Running Commands

Preventing long-running commands is critical since Redis handles all operations on a single thread.

- **Avoid full key scans**: Never use KEYS * or SMEMBERS on large sets in production. Use SCAN instead for incremental iteration.
- **Limit Lua script duration**: Break complex scripts into smaller steps and test for performance in staging.
- **Use pipelining**: Send multiple commands in one round-trip to reduce overall time spent per operation.
- **Limit list and set access**: Use ranges or batch operations for large data structures.

```
LRANGE mylist 0 99   # Good
LRANGE mylist 0 -1   # Risky on large lists
```

- **Enable eviction policies**: To avoid OOM errors that can freeze Redis, enable LRU or LFU eviction:

```
CONFIG SET maxmemory-policy allkeys-lru
```

- **Monitor regularly**: Use CLIENT LIST, SLOWLOG, and LATENCY in combination to detect problematic patterns early.

Preventing Full Disk Issues

Running out of disk space in a Redis environment can lead to failed writes, snapshot errors, and service unavailability. Redis relies on disk storage for persistence (RDB and AOF files), temporary dumps, and logs especially when persistence is enabled. On platforms like Elestio, while the infrastructure is managed, users are responsible for monitoring disk usage, configuring retention policies, and managing backups. This guide covers how to monitor disk consumption, configure alerts, remove unused data, and follow best practices to prevent full disk scenarios in a Redis setup using Docker Compose.

Monitoring Disk Usage

Disk usage monitoring is essential for spotting unusual growth before it leads to failures. In Docker Compose setups, you'll need both host-level and container-level visibility.

Inspect the host system storage

Run this on the host machine to check which mount point is filling up:

```
df -h
```

This shows available and used space for each volume. Identify the mount point used by your Redis volume—usually mapped to something like /var/lib/docker/volumes/redis_data/_data.

Check disk usage from inside the container

Open a shell inside the Redis container:

```
docker-compose exec redis sh
```

Inside the container, check the data directory size:

```
du -sh /data
```

This reveals total usage by persistence files (appendonly.aof, dump.rdb, temporary files). You can inspect individual file sizes with:

```
ls -lh /data
```

Configuring Alerts and Cleaning Up Storage

Monitoring alone isn’t enough—automated alerts and safe cleanup prevent downtime. You can inspect disk usage across Docker resources on the host with:

```
docker system df
```

Identify unused Docker volumes

```
docker volume ls
```

To remove a specific unused volume:

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

> **Warning**: Never delete the volume mapped to your Redis data unless you’ve backed up its contents and confirmed it is not in use.

Trigger AOF file compaction

If AOF persistence is enabled, the append-only file can grow large over time. You can manually trigger a rewrite to compact the file:

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

This creates a smaller AOF file containing the same dataset.

Clean up old snapshots

If you are using RDB snapshots, they’re stored in /data within the container (mapped to a host volume). To clean up, list them first:

```
docker-compose exec redis ls -lh /data
```

Remove unnecessary .rdb files with:

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

Managing & Optimizing Temporary Files

Redis creates temporary files during fork operations for AOF rewrites and RDB saves. These are stored in the container’s /tmp directory.

Monitor temporary file usage:

```
docker-compose exec redis du -sh /tmp
```

If /tmp fills up, writes and forks may fail. You can change the temporary directory by modifying the dir directive in redis.conf to point to /data, which is volume-backed:

```
dir /data
```

Restart the container to apply changes.

Best Practices for Disk Space Management

Long-term disk space health in Redis requires proactive design and ongoing management.

- **Avoid storing binary blobs**: Store large files (images, PDFs, etc.) outside Redis and use Redis only for keys/metadata. Use object storage for large content.
- **Disable persistence if not needed**: For ephemeral cache use cases, you can disable persistence entirely to reduce disk usage:

```
appendonly no
save ""
```

- **Limit AOF growth**: Fine-tune AOF rewrite behavior in redis.conf:

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

- **Rotate logs in containers**: If logging to file (e.g., /var/log/redis/redis-server.log), configure logrotate on the host or use Docker log rotation options via docker-compose.yml :

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

- **Evict old keys with TTLs**: Set expiration on cache keys to prevent unbounded growth:

```
SET session:<id> "data" EX 3600
```

- **Monitor data size**: Use INFO persistence and INFO memory to track memory usage and AOF file size:

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

- **Offload backups**: Backups stored in /data should be moved off the container host. Use Elestio backup tools or mount a remote backup volume in your docker-compose.yml.

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.