# Identifying Slow Queries

Slow commands can impact KeyDB performance, especially under high concurrency or when inefficient data access patterns are used. Whether you're running KeyDB on Elestio via the dashboard, inside a Docker Compose setup, or accessing it through the CLI, KeyDB includes powerful introspection tools like the slow log and latency tracking.

This guide shows how to detect slow operations using KeyDB's built-in slowlog, analyze latency issues, and optimize performance through configuration tuning and query best practices.

# Inspecting Slow Commands from the Terminal

KeyDB supports the Redis-compatible SLOWLOG feature to record commands that exceed a configured execution time threshold. These logs are useful to spot expensive operations and server stalls.

## Connect to Your KeyDB Instance via Terminal

Use keydb-cli or redis-cli to connect to your KeyDB instance:

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

Replace <host>, <port>, and <password> with the credentials available in your Elestio dashboard.

## View the Slowlog Threshold

Check what execution time (in microseconds) is considered "slow":

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

The default is 10000 (10 milliseconds). Commands slower than this will be logged.

## View the Slow Query Log

To retrieve recent slow operations:

```
SLOWLOG GET 10
```

This shows the 10 most recent slowlog entries, each with:

- The command that was executed
- The timestamp
- Execution duration in microseconds
- Any arguments passed to the command

# Analyzing Inside Docker Compose

If you’re running KeyDB via Docker Compose, you can inspect slow queries from within the container environment.

## Access the KeyDB Container

Launch a shell in your container:

```
docker-compose exec keydb bash
```

Connect to KeyDB using:

```
keydb-cli -a $KEYDB_PASSWORD
```

Ensure that REDIS_PASSWORD (or KEYDB_PASSWORD) is defined in your .env file or Compose environment variables.

## Adjust Slowlog Settings

You can view or modify the slowlog threshold dynamically:

```
CONFIG SET slowlog-log-slower-than 5000
```

This temporarily changes the threshold to 5 milliseconds, which is useful for debugging under lower latency conditions.

## Increase the Number of Stored Entries

Check how many slowlog entries are retained:

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

To store more slowlog entries:

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

This helps in long-running investigations or during load testing.

# Using the Latency Monitoring Feature

KeyDB inherits Redis's latency monitoring tools, providing additional insights beyond command duration as fork stalls, I/O blocks, or memory pressure.

## Enable Latency Monitoring

Latency tracking is often enabled by default. Run:

```
LATENCY DOCTOR
```

This provides a high-level diagnostic report of system latency spikes and potential root causes, including slow commands, AOF rewrites, and blocking operations.

## View Latency History for Specific Events

Track the latency of specific operations like:

```
LATENCY HISTORY command
```

Other event categories include:

- fork – Background save or AOF rewrite delays
- aof-write – Append-only file sync lag
- command – General command execution delays

# Understanding and Resolving Common Bottlenecks

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

## Common Causes of Slowness

- **Large keys**: Commands like LRANGE, SMEMBERS, or HGETALL on large datasets.
- **Blocking commands**: Such as BLPOP, BRPOP, or long-running Lua scripts.
- **Forking delays**: Caused by SAVE or AOF background rewriting.

## Best Practices for Performance

- **Use SCAN** instead of KEYS to iterate large keyspaces safely.
- **Limit range queries**: Use LRANGE 0 99 instead of fetching full lists.
- **Enable pipelining**: Reduce round trips by batching commands.
- **Avoid multi-key ops**: Especially in clustered deployments, where they can cause performance issues or fail.

# Optimizing with Configuration Changes

KeyDB performance can be significantly tuned by adjusting memory and persistence-related settings.

## Common Tuning Examples

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

These adjust eviction and persistence behaviours. Use these with caution:

- Disabling RDB/AOF improves speed but removes durability.
- LRU/TTL policies control memory usage under load.

---

Revision #1
Created 2025-06-25 09:14:07 UTC
Updated 2025-06-25 09:19:59 UTC