# 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.

---

Revision #3
Created 2025-05-20 10:24:38 UTC
Updated 2025-05-20 10:28:39 UTC