



## Redis latency monitoring framework

Redis is often used in the context of demanding use cases, where it serves a large number of queries per second per instance, and at the same time, there are very strict latency requirements both for the average response time and for the worst case latency.

While Redis is an in-memory system, it deals with the operating system in different ways, for example, in the context of persisting to disk. Moreover Redis implements a rich set of commands. Certain commands are fast and run in constant or logarithmic time, other commands are slower  $O(N)$  commands that can cause latency spikes.

Finally Redis is single threaded: this is usually an advantage from the point of view of the amount of work it can perform per core, and in the latency figures it is able to provide, but at the same time it poses a challenge from the point of view of latency, since the single thread must be able to perform certain tasks incrementally, for example key expiration, in a way that does not impact the other clients that are served.

For all these reasons, Redis 2.8.13 introduced a new feature called **Latency Monitoring**, that helps the user to check and troubleshoot possible latency problems. Latency monitoring is composed of the following conceptual parts:

- Latency hooks that sample different latency sensitive code paths.
- Time series recording of latency spikes split by different event.
- Reporting engine to fetch raw data from the time series.
- Analysis engine to provide human readable reports and hints according to the measurements.

The remaining part of this document covers the latency monitoring subsystem details, however for more information about the general topic of Redis and latency, please read the [Redis latency problems troubleshooting](#) page in this documentation.

### Events and time series

Different monitored code paths have different names, and are called *events*. For example `command` is an event measuring latency spikes of possibly slow commands executions, while `fast-command` is the event name for the monitoring of the  $O(1)$  and  $O(\log N)$  commands. Other events are less generic, and monitor a very specific operation performed by Redis. For example the `fork` event only monitors the time taken by Redis to execute the `fork(2)` system call.

A latency spike is an event that runs in more time than the configured latency threshold. There is a separated time series associated with every monitored event. This is how the time series work:

- Every time a latency spike happens, it is logged in the appropriate time series.
- Every time series is composed of 160 elements.
- Each element is a pair: a Unix timestamp of the time the latency spike was measured, and the number of milliseconds the event took to executed.
- Latency spikes for the same event happening in the same second are merged (by taking the maximum latency), so even if continuous latency spikes are measured for a given event, for example because the user set a very low threshold, at least 180 seconds of history are available.
- For every element the all-time maximum latency is recorded.

The framework monitors and logs latency spikes in the execution time of these events:

- `command`: regular commands.
- `fast-command`:  $O(1)$  and  $O(\log N)$  commands.
- `fork`: the `fork(2)` system call.
- `rdb-unlink-temp-file`: the `unlink(2)` system call.
- `aof-write`: writing to the AOF - a `catchall` event `fsync(2)` system calls.
- `aof-fsync-always`: the `fsync(2)` system call when invoked by the `appendfsync always` policy.
- `aof-write-pending-fsync`: the `fsync(2)` system call when there are pending writes.
- `aof-write-active-child`: the `fsync(2)` system call when performed by a child process.
- `aof-write-alone`: the `fsync(2)` system call when performed by the main process.
- `aof-fstat`: the `fstat(2)` system call.
- `aof-rename`: the `rename(2)` system call for renaming the temporary file after completing [BGREWRITEAOF](#).
- `aof-rewrite-diff-write`: writing the differences accumulated while performing [BGREWRITEAOF](#).
- `active-defrag-cycle`: the active defragmentation cycle.
- `expire-cycle`: the expiration cycle.
- `eviction-cycle`: the eviction cycle.
- `eviction-del`: deletes during the eviction cycle.

## How to enable latency monitoring

What is high latency for one use case is not high latency for another. There are applications where all the queries must be served in less than 1 millisecond and applications where from time to time a small percentage of clients experiencing a 2 second latency is acceptable.

So the first step to enable the latency monitor is to set a **latency threshold** in milliseconds. Only events that will take more than the specified threshold will be logged as latency spikes. The user should set the threshold according to their needs. For example if for the

requirements of the application based on Redis the maximum acceptable latency is 100 milliseconds, the threshold should be set to such a value in order to log all the events blocking the server for a time equal or greater to 100 milliseconds.

The latency monitor can easily be enabled at runtime in a production server with the following command:

```
CONFIG SET latency-monitor-threshold 100
```

By default monitoring is disabled (threshold set to 0), even if the actual cost of latency monitoring is near zero. However while the memory requirements of latency monitoring are very small, there is no good reason to raise the baseline memory usage of a Redis instance that is working well.

## Information reporting with the LATENCY command


The user interface to the latency monitoring subsystem is the **LATENCY** command. Like many other Redis commands, **LATENCY** accepts subcommands that modifies its behavior. These subcommands are:

- **LATENCY LATEST** - returns the latest latency samples for all events.
- **LATENCY HISTORY** - returns latency time series for a given event.
- **LATENCY RESET** - resets latency time series data for one or more events.
- **LATENCY GRAPH** - renders an ASCII-art graph of an event's latency samples.
- **LATENCY DOCTOR** - replies with a human-readable latency analysis report.

Refer to each subcommand's documentation page for further information.

---

This website is open source software. See all credits.

Sponsored by  **redislabs**  
HOME OF REDIS