Memcached In-memory cache

CONCEPTS

Memcached is a plain in-memory key-value cache: no persistence, no auth by default, and old items are evicted when memory is full. Never expose it to the internet.

Client hashes key to a server
↓ `get <key>`
memcached slabs + LRU
↓ `miss`
Database source of truth
↓ `set <key>`
memcached next read hits

Servers know nothing about each other: the client library picks the server per key, so adding nodes adds capacity.

A ttl of 0 means no expiry, but items can still be evicted; a ttl above 30 days is read as a Unix timestamp.

RUN THE DAEMON

`memcached -m <MB>`
Set item memory (default 64)

`memcached -p <port>`
TCP port (default 11211)

`memcached -l <addr>`
Listen on one interface

`memcached -d -u <user>`
Daemonize, drop to a user

`memcached -c <n>`
Max connections (default 1024)

`memcached -t <n>`
Worker threads (default 4)

`memcached -v / -vv`
Log errors / also commands

`memcached -s <path>`
Listen on a UNIX socket

`memcached -U <port>`
Enable UDP (off by default)

`memcached -M`
Error on full instead of evicting

`memcached -I <size>`
Max item size (default 1m)

`memcached -P <file>`
Write a pid file (with -d)

`memcached -o <opt>,<opt>`
Extended options

RUN THE DAEMON (cont.)

`memcached -V`
Print version

TALK TO IT

`nc <host> <port>`
Open a raw text session

`printf '<cmd>\r\n' | nc -q1 <host> <port>`
Send one command

`telnet <host> <port>`
Same, with telnet

`version`
Show server version

`quit`
Close the session

STORE AND FETCH

`set <key> <flags> <ttl> <bytes>`
Store, overwrite if present

`get <key> [<key> ...]`
Fetch one or more keys

`delete <key>`
Remove a key

`add <key> <flags> <ttl> <bytes>`
Store only if absent

`replace <key> <flags> <ttl> <bytes>`
Store only if present

`append <key> 0 0 <bytes>`
Add bytes after the value

`gets <key>`
Fetch with a CAS id

`cas <key> <flags> <ttl> <bytes> <cas>`
Store only if unchanged

`touch <key> <ttl>`
Update expiry only

`gat <ttl> <key>`
Get and set new expiry

COUNTERS

`incr <key> <n>`
Add n to a number

`decr <key> <n>`
Subtract n, floor at 0

`set <key> 0 0 1`
Seed a counter with 0

META COMMANDS

`mg <key> v`
Meta get: return value

`mg <key> v t f c k`
Value plus ttl, flags, CAS, key

`ms <key> <bytes> T<ttl>`
Meta set with a ttl

`md <key>`
Meta delete

STATS AND INSPECTION

`stats`
General counters

`stats settings`
Effective config of the daemon

`stats items`
Per-slab item counts and age

`stats slabs`
Per-slab memory use

`stats reset`
Zero resettable counters

`lru_crawler metadump all`
List keys with metadata

`watch`
Stream live log of requests

`flush_all [<delay>]`
Invalidate all items

KEY STATS TO READ

Fields in stats

Field	Look for
`get_hits / get_misses`	hits / (hits+misses)
`evictions`	rising: too little RAM
`curr_items`	items now
`bytes / limit_maxbytes`	used vs. limit
`curr_connections`	clients now
`listen_disabled_num`	non-zero: -c hit

RECIPES

Round trip from the shell

```
printf 'set k 0 60 5\r\nhello\r\nget k\r\nquit\r\n' | nc -q1 127.0.0.1 11211
```

Hit ratio

```
printf 'stats\r\nquit\r\n' | nc -q1 127.0.0.1 11211 |
  awk '/get_hits/{h=$3} /get_misses/{m=$3} END{print h/(h+m)}'
```