wrk HTTP load tester

START HERE

`wrk -t2 -c20 -d10s <url>`
Quick 10 s test

`wrk -c<N> -t<N> -d<time>`
The three core knobs

`wrk -t12 -c400 -d30s <url>`
Heavier load

`wrk -s <script.lua> <url> -- <args>`
Pass script args

`wrk --latency <url>`
Percentile latencies

`wrk -H "<header>" <url>`
Add a header

`wrk -s <script.lua> <url>`
Run a Lua script

`wrk --timeout 5s <url>`
Count slow replies

setup once per thread → **init** per thread, gets args → **loop** **request / response** every call → **end** **done** final summary

How a script hooks into a run

CONCEPTS

Load test only systems you own or have written permission to test. wrk can take a server down.

Connections are split across threads (connections / threads each), so -c must be at least -t. Use as many threads as cores, no more.

wrk reports requests/sec; for constant-rate load and coordinated-omission-free latency use the wrk2 fork (-R rate).

RUNNING A TEST

`wrk -t<n> -c<n> -d<time> <url>`
Run a benchmark

`-t, --threads <n>`
Worker threads

RUNNING A TEST (cont.)

`-c, --connections <n>`
Open connections in total

`-d, --duration <time>`
Test length (2s, 2m, 2h)

`--latency`
Print latency distribution

`--timeout <time>`
Timeout per response

`-H, --header "<name: value>"`
Add a request header

`-s, --script <file>`
Load a Lua script

`-- <arg>...`
Pass args to init(args)

`-v, --version`
Show version

Raise open-file and ephemeral-port limits (ulimit -n) on the client, and keep the server's listen backlog above -c.

READING OUTPUT

Output lines

Line	Meaning
Latency	Per-request time: avg, stdev, max
Req/Sec	Per-thread rate, not total
Requests/sec	Total throughput
Transfer/sec	Bytes read per second
Socket errors	connect/read/write/timeout counts
Non-2xx or 3xx	Responses with error status

Check for socket errors and non-2xx counts first: fast errors inflate requests/sec.

With --latency read the 99% line, not the average.

LUA SCRIPTING

`wrk.method = "POST"`
Set the HTTP method

`wrk.body = '<data>'`
Set the request body

LUA SCRIPTING (cont.)

`wrk.headers["<name>"] = "<value>"`
Set a header

`wrk.path / wrk.host / wrk.port`
Target parts (read)

`function request()`
Return the next request

`wrk.format(method, path, headers, body)`
Build a request string

`function response(status, headers, body)`
Inspect each response

`function init(args)`
Per-thread setup

`function setup(thread)`
Run once per thread

`function done(summary, latency, requests)`
Custom final report

`function delay()`
Return ms to wait

LUA SCRIPTING (cont.)

`latency:percentile(99.0)`
Latency percentile in done()

`summary.errors.timeout`
Error counters in done()

`thread:set("<k>", v) / thread:get("<k>")`
Share data with threads

Static changes (method, path, headers, body) cost nothing. Building a request per call or using response() lowers the load wrk can generate.

RECIPES

Basic latency run

```
wrk -t4 -c100 -d30s --latency
http://localhost:8080/
```

RECIPES (cont.)

POST JSON (post.lua)

```
wrk.method = "POST"
wrk.body   = '{"name":"x"}'
wrk.headers["Content-Type"] =
"application/json"

# wrk -t2 -c20 -d10s -s
post.lua http://localhost:8080/
items
```

Vary the path per request

```
counter = 0
request = function()
  counter = counter + 1
  return wrk.format("GET", "/
item/" .. counter)
end
```

Print status errors

```
response = function(status,
headers, body)
  if status ~= 200 then
print(status) end
end
```