ApacheBench HTTP load tester

START HERE

ab -n 1000 -c 10 *<url>*
1000 requests, 10 at once

ab -k -n 5000 -c 50 *<url>*
Reuse connections

ab -t 30 -c 20 *<url>*
Run for 30 seconds

ab -H "*<header>***"** *<url>*
Add a header

ab -p *<file>* **-T** *<type>* *<url>*
POST a file

ab -A *<user>*:*<pass>* *<url>*
Basic auth

ab -r -n 1000 -c 100 *<url>*
Survive socket errors

ab -e *<file.csv>* *<url>*
Export percentiles

CONCEPTS

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

The URL needs a path: use http://host/ with the trailing slash, not http://host.

-c clients open connections
↓ send
Server one URL, same request
↓ reply
-n requests counted until done
↓ measure
Report rps, time, percentiles
What ab does in a run

ab is single-threaded and HTTP/1.x only, so it saturates one core early. For more load or HTTP/2 use wrk or k6.

Fixed buffers and partial HTTP handling are documented limits: treat results as relative, not absolute.

RUNNING A TEST

ab [options] *<url>*
Benchmark one URL

-n *<requests>*
Total requests (default 1)

-c *<concurrency>*
Parallel requests (default 1)

-t *<seconds>*
Time limit; implies -n 50000

-k
Enable HTTP KeepAlive

-s *<seconds>*
Socket timeout (default 30)

RUNNING A TEST (cont.)

-r
Continue after socket errors

-l
Ignore varying response length

-i
Send HEAD instead of GET

-m *<method>*
Custom HTTP method
Needs ab from httpd 2.4.10 or newer

-B *<address>*
Bind to a local address

-b *<bytes>*
TCP send/receive buffer size

-V
Show version

-h
Show usage

-n defaults to 1, which tells you nothing. Keep -c no larger than -n.

REQUESTS

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

-C *<name>*=*<value>*
Add a cookie (repeatable)

-p *<file>* **-T** *<type>*
POST the file's contents

-u *<file>* **-T** *<type>*
PUT the file's contents

-T *<content-type>*
Content-Type for POST/PUT

-A *<user>*:*<pass>*
Send Basic auth credentials

-X *<proxy>***[:***<port>***]**
Go through a proxy

-P *<user>*:*<pass>*
Proxy Basic auth

REQUESTS (cont.)

-p sends the file verbatim as the body; -T defaults to text/plain, so set it for JSON or forms.

TLS

ab https://*<host>***/**
Benchmark over TLS

-f *<protocol>*
Force TLS1.2, TLS1.1, TLS1 or ALL

-Z *<ciphersuite>*
Choose a cipher suite

-E *<file.pem>*
Client certificate and key
Needs httpd 2.4.36 or newer

Without -k every request pays a full TLS handshake; that is what you measure.

READING OUTPUT

Key output lines

Line	Meaning
Requests per second	Total throughput (mean)
Time per request (1st)	Mean latency seen by one client
Time per request (2nd)	Mean across all concurrent clients
Failed requests	Connect, read, length errors
Non-2xx responses	Status outside 200-299
Transfer rate	KB/s received
Percentage served	Latency percentiles in ms

READING OUTPUT (cont.)

Check Failed requests and Non-2xx first: fast error pages inflate requests per second.

Read the 95% and 99% rows of the percentile table, not the mean. 100% is the slowest request.

-e *<file.csv>*
CSV of percentile vs ms

-g *<file.tsv>*
TSV of every request (gnuplot)

-q
Hide progress output

-v *<level>*
Verbosity (2 info, 3 codes, 4 headers)

-w
Print results as HTML tables

-d
Hide percentile table

-S
Hide median and stddev

RECIPES

Quick smoke test

```
ab -n 1000 -c 10 -k http://
localhost:8080/
```

POST JSON

```
echo '{"name":"x"}' > body.json
ab -n 500 -c 10 -p body.json -T
application/json http://
localhost:8080/items
```

Authenticated API call

```
ab -n 200 -c 5 -H
"Authorization: Bearer <token>"
https://example.test/api/me
```

Plot latency

```
ab -n 2000 -c 20 -g out.tsv
http://localhost:8080/
# gnuplot: plot "out.tsv" using
9 smooth sbezier with lines
```