k6 Load testing

START HERE

`k6 new`
Create a starter script

`k6 run -e <NAME>=<value> script.js`
Pass a variable

`k6 run script.js`
Run a load test

`k6 run --out json=<file> script.js`
Save raw metrics

`k6 run --vus 10 --duration 30s script.js`
Set load on the CLI

`k6 run --http-debug script.js`
Print requests

`k6 run --iterations 1 script.js`
Smoke test once

`k6 cloud run script.js`
Run in Grafana Cloud

init imports, options → once → **setup()** returns data → data → **default()** loops per VU → data → **teardown()** cleanup

CONCEPTS

Scripts are JavaScript, but k6 runs them in its own Go runtime: no Node.js APIs, no npm install.

Options resolve in this order: defaults, script options, environment variables, then CLI flags (highest).

Failed checks do not change the exit code. Add a threshold on the checks metric to fail the run.

RUNNING

`k6 new [<file>]`
Scaffold a script

`k6 run <script>`
Run a script locally

`k6 run -`
Read the script from stdin

`k6 run <archive.tar>`
Run a script archive

`--vus <n>` / `-u <n>`
Concurrent virtual users

`--duration <time>` / `-d <time>`
Total run time, e.g. 3m

`--iterations <n>` / `-i <n>`
Total iterations across VUs

`--stage <time>:<target>`
Ramp VUs, repeatable (-s)
Example: -s 30s:20 -s 1m:20 -s 10s:0. Equivalent to options.stages.

`--once`
One iteration with one VU

`--scenario <name>`
Run only the named scenarios

`--linger / -l`
Keep running after the test ends

`--no-setup / --no-teardown`
Skip setup() or teardown()

VARIABLES AND TAGS

`-e <NAME>=<value>` / `--env <NAME>=<value>`
Set a variable in __ENV

`--include-system-env-vars`
Expose real env vars to __ENV

`--tag <name>=<value>`
Tag every metric of the run

`--config <file>` / `-c <file>`
Read options from a JSON file

VARIABLES AND TAGS (cont.)

`K6_<OPTION>`
Set most options by env var
For example K6_VUS=10 K6_DURATION=30s k6 run script.js.

OUTPUT AND SUMMARY

`--out json=<file>` / `-o json=<file>`
Write every metric point as JSON

`--out csv=<file>`
Write metric points as CSV

`--out <name>=<target>`
Stream to InfluxDB or another output
Repeat -o to send to several outputs at once.

`--summary-export <file>`
Write the end summary as JSON
Discouraged by the docs; prefer handleSummary().

`--summary-mode <mode>`
compact, full or disabled

`--summary-trend-stats <list>`
Choose trend stats, e.g. avg,p(95),max

`--quiet / -q`
Hide the progress bar

`--verbose / -v`
Verbose logging

`--no-color`
Plain output for logs and CI

DEBUGGING

`--http-debug`
Log HTTP requests and responses

`--http-debug=full`
Include request and response bodies

`--no-thresholds`
Run without evaluating thresholds

`--throw / -w`
Throw on failed requests

`--insecure-skip-tls-verify`
Accept any TLS certificate

`--no-connection-reuse`
Disable keep-alive

`--discard-response-bodies`
Drop bodies to save memory

SCRIPT OPTIONS

options block

```
export const options = {
  vus: 10,
  duration: '30s',
};
```

Stages

```
stages: [
  { duration: '30s', target: 20 },
  { duration: '1m', target: 20 },
  { duration: '10s', target: 0 },
],
```

Thresholds

```
thresholds: {
  http_req_duration: ['p(95)<200'],
  http_req_failed: ['rate<0.01'],
  checks: ['rate>0.99'],
},
```

Abort on failure

```
http_req_duration: [{
  threshold: 'p(99)<500',
  abortOnFail: true,
  delayAbortEval: '10s',
}],
```

Scenario

```
scenarios: {
  api: {
    executor: 'constant-arrival-rate',
    rate: 20, timeUnit: '1s',
    duration: '1m',
    preAllocatedVUs: 10, maxVUs: 50,
  },
},
```

Executors

Name	Drives load by
`constant-vus`	fixed VU count
`ramping-vus`	VU stages
`shared-iterations`	iterations split across VUs
`per-vu-iterations`	iterations per VU
`constant-arrival-rate`	fixed iteration rate
`ramping-arrival-rate`	iteration rate stages

Threshold syntax

Metric type	Aggregations
`Counter`	count, rate
`Gauge`	value
`Rate`	rate
`Trend`	avg, min, max, med, p(N)

A threshold key can filter by tag: 'http_req_duration{type:API}': ['p(95)<500'].

SCRIPT API

Minimal test

```
import http from 'k6/http';
import { check, sleep } from 'k6';

export default function () {
  const res = http.get('https://test.k6.io/');
  check(res, { 'status is 200': (r) => r.status === 200 });
  sleep(1);
}
```

`check(<val>, { <name>: <fn> })`
Assert without stopping the test

`sleep(<seconds>)`
Pause the VU between iterations

`group('<name>', <fn>)`
Group requests under a name

`http.get(<url>)`
GET request

`http.post(<url>, <body>, <params>)`
POST request

`__ENV.<NAME>`
Read a variable set with -e

`export function setup()`
Run once; return shared data

`export function teardown(data)`
Run once after the test

`export function handleSummary(data)`
Customize the end report

ARCHIVE AND CLOUD

`k6 archive <script>`
Bundle script and files in a tar

`k6 cloud run <script>`
Run on Grafana Cloud k6

RECIPES

Quick smoke test

```
k6 run --vus 1 --iterations 1 script.js
```

Ramp, hold, ramp down

```
k6 run -s 30s:50 -s 5m:50 -s 30s:0 script.js
```

Point one script at several hosts

```
k6 run -e BASE_URL=https://staging.example.com script.js
# in the script: http.get(`${__ENV.BASE_URL}/health`)
```

Debug a failing request

```
k6 run --http-debug=full --iterations 1 --vus 1 script.js
```