k6 Load testing

CONCEPTS

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

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

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

--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)

VARIABLES AND TAGS

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

OUTPUT AND SUMMARY

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

DEBUGGING

--http-debug
Log HTTP requests and responses

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

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'],
},
```

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

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

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