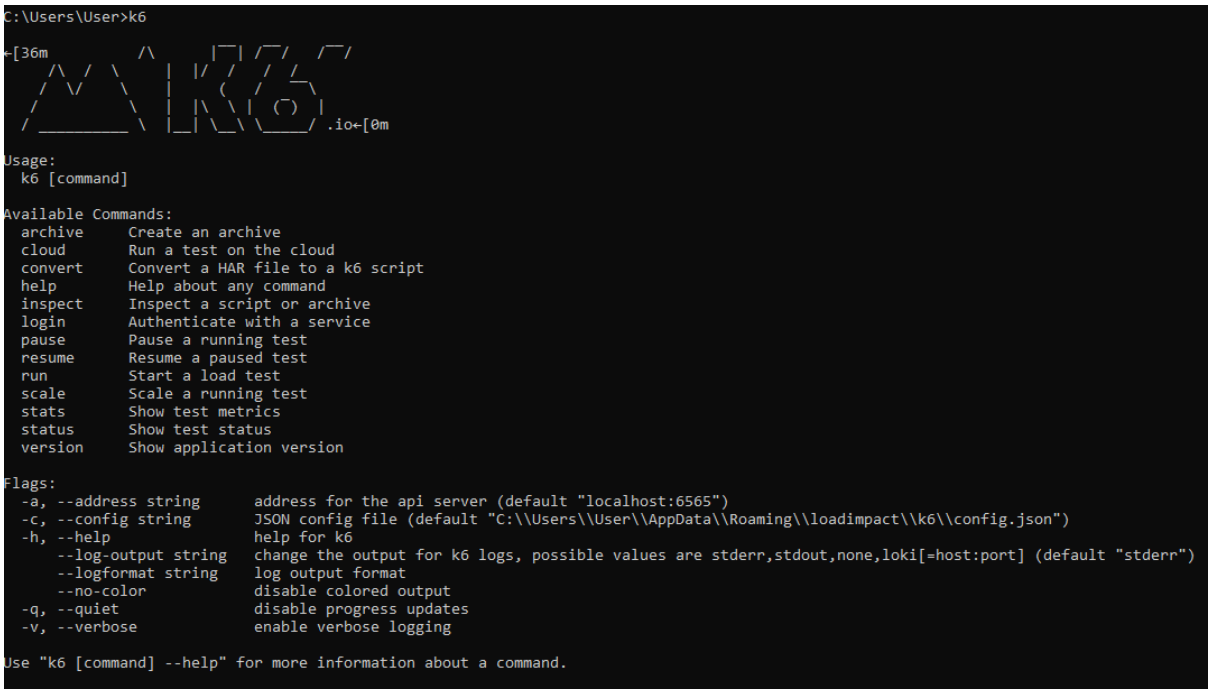
# K6 Overview

## Installation

1. Download and install the latest .msi package.
2. Open command prompt and run: `k6`

If installation was successful, short k6 manual should be displayed:




## Writing a simple k6 script

1. Create `google_search.js` file.
2. Paste the following code:

```
import http from 'k6/http';

export default function () {
  const search_query = 'k6 installing';
  const search_url = `https://www.google.com/search?q=${search_query}`;

  const response = http.get('https://test.k6.io');
  console.log(response.body);
}
```

3. Open command prompt and run `k6 run google_search.js`

This script will execute search request in google. It terminal output you should see HTML response code, that will contain search results.

## Using checks

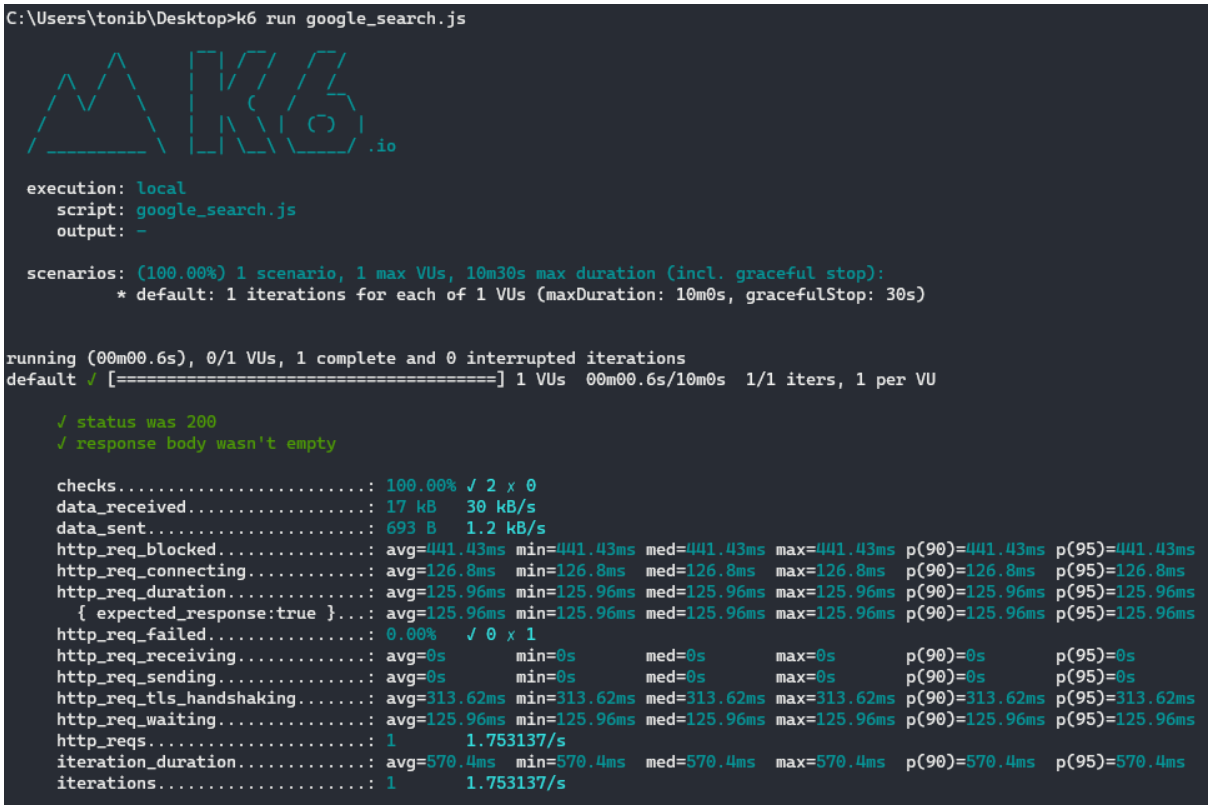
1. Update `google_search.js` with the following code:

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

function check_google_response(res) {
  check(res, {
    'status was 200': (r) => r.status == 200,
    'response body wasn\'t empty': (r) => r.body.length > 0,
  });
}

export default function () {
  const search_query = 'k6 installing';
  const search_url = `https://www.google.com/search?q=${search_query}`;

  const response = http.get('https://test.k6.io');
  check_google_response(response);
}
```

It this code we added a function that will check response status and body length.

2. Open command prompt and run `k6 run google_search.js`

You should see checks result in terminal output.

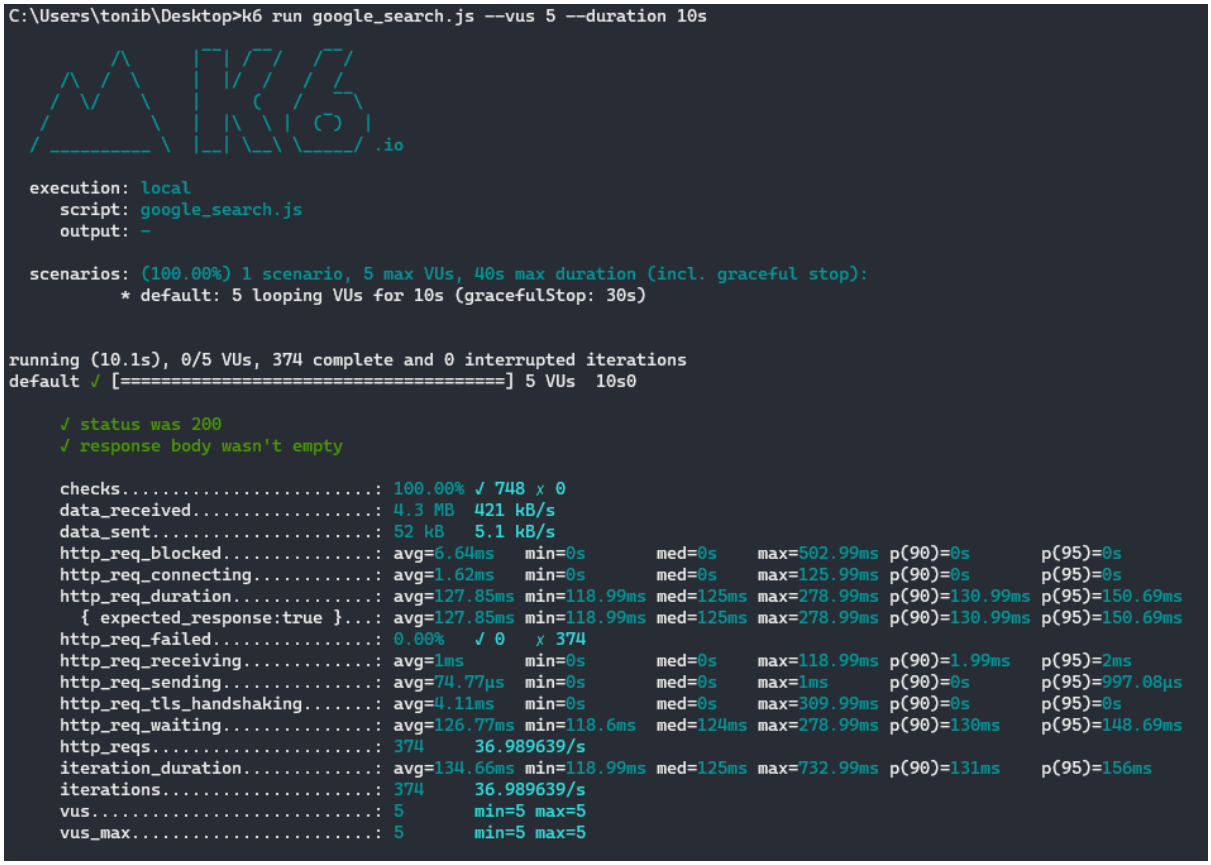



## Using options

If we want to implement a load testing, we should use VUs (Virtual Users), which are users that will execute requests in parallel.

1. Open command prompt and run `k6 run google_search.js --vus 5 --duration 10s`

This options will create 5 virtual users, which will execute requests for 10 seconds.




It the output, you can see that 374 requests were executed in total.

If you want to avoid adding `--vus 5 --duration 10s` each time, you can include these options into script:

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

export let options = {
  vus: 5,
  duration: '10s',
};

function check_google_response(res) {
  check(res, {
    'status was 200': (r) => r.status == 200,
    'response body wasn\'t empty': (r) => r.body.length > 0,
  });
}

export default function () {
  const search_query = 'k6 installing';
  const search_url = `https://www.google.com/search?q=${search_query}`;

    const response = http.get('https://test.k6.io');
```

```
    check_google_response(response);
}
```

## Ramping behavior using stages

You can also configure ramping behavior by adding stages option.

1. Update `google_search.js` with the following code:

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

export let options = {
  stages: [
    { duration: '5s', target: 5 },
    { duration: '10s', target: 10 },
    { duration: '5s', target: 5 },
  ],
};

function check_google_response(res) {
  check(res, {
    'status was 200': (r) => r.status == 200,
    'response body wasn\'t empty': (r) => r.body.length > 0,
  });
}

export default function () {
  const search_query = 'k6 installing';
  const search_url = `https://www.google.com/search?q=${search_query}`;

    const response = http.get('https://test.k6.io');
    check_google_response(response);
}
```

2. Open command prompt and run `k6 run google_search.js`

In this case, test running time will be 20 seconds divided into three stages:

1. 0-5s - will gradually increase the number of VUs from 0 to 5
2. 5-15 - will gradually increase the number of VUs from 5 to 10
3. 15-20 - will gradually reduce the number of VUs from 10 to 5