Complete K6 Load Testing Tutorial

Table of Contents

---

1. Introduction to K6 {#introduction}

K6 is a modern load testing tool designed for developers and testers to easily create and run performance tests. Unlike traditional tools, K6 uses JavaScript (ES6) for test scripting, making it accessible to web developers.

Key Features:

- **Developer-friendly**: Uses JavaScript for test scripts
- **CLI-based**: Runs from command line, perfect for CI/CD
- **High performance**: Written in Go, can handle thousands of virtual users
- **Rich metrics**: Provides detailed performance metrics
- **Extensible**: Supports custom metrics and outputs

When to Use K6:

- API performance testing
- Website load testing
- Microservices testing
- CI/CD pipeline integration
- Stress testing and capacity planning

---

2. Installation {#installation}

Method 1: Using Package Managers

Windows (Chocolatey):

```bash
choco install k6
```

macOS (Homebrew):

```bash
brew install k6
```

Linux (APT):

```bash
sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv-keys C5AD17C747E3415A3642D57D77C6C491D6
echo "deb https://dl.k6.io/deb stable main" | sudo tee /etc/apt/sources.list.d/k6.list
sudo apt-get update
sudo apt-get install k6
```

Method 2: Direct Download

1. Visit https://k6.io/docs/getting-started/installation/
2. Download the binary for your OS
3. Extract and add to PATH

Method 3: Docker

```bash
docker run --rm -i grafana/k6:latest run - <script.js
```

Verify Installation

```bash
k6 version
```

---

3. Basic Concepts {#basic-concepts}

Virtual Users (VUs)

Virtual Users simulate real users interacting with your application. Each VU runs your test script independently.

Iterations

An iteration is one complete execution of your test script by a VU.

Scenarios

Scenarios define how your test should behave - how many VUs, for how long, with what pattern.

Metrics

K6 automatically collects various metrics:

- **Response Time**: Time taken for requests
- **Throughput**: Requests per second
- **Error Rate**: Percentage of failed requests
- **Data Transfer**: Bytes sent/received

Checks

Checks are like assertions - they verify that responses meet certain criteria.

Thresholds

Thresholds define pass/fail criteria for your test based on metrics.

---

4. Your First K6 Test {#first-test}

Let's create a simple test to understand K6 basics.

Step 1: Create Your First Test Script

Create a file called `first-test.js`:

javascript

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

export default function () {
  // Make an HTTP GET request
  let response = http.get('https://httpbin.org/get');

  // Check that response is successful
  check(response, {
    'status is 200': (r) => r.status === 200,
    'response time < 500ms': (r) => r.timings.duration < 500,
  });

  // Wait 1 second between iterations
  sleep(1);
}
```

Step 2: Run Your Test

bash

```bash
k6 run first-test.js
```

Step 3: Understanding the Output

K6 will show:

- Test progress in real-time
- Summary of metrics at the end
- Any failed checks

Step 4: Customize Test Options

Add options to your test:

javascript

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

export let options = {
  vus: 10,      // 10 virtual users
  duration: '30s', // Run for 30 seconds
};

export default function () {
  let response = http.get('https://httpbin.org/get');

  check(response, {
    'status is 200': (r) => r.status === 200,
  });

  sleep(1);
}
```

---

5. Test Structure and Lifecycle {#test-structure}

Test Lifecycle Functions

K6 provides four lifecycle functions:

javascript

```javascript
// 1. Setup - runs once before test starts
export function setup() {
  console.log('Setting up test data...');
  return { token: 'abc123' };
}

// 2. Default - main test function (runs for each VU iteration)
export default function (data) {
  console.log('Token from setup:', data.token);
  // Your test logic here
}

// 3. Teardown - runs once after test ends
export function teardown(data) {
  console.log('Cleaning up...');
}

// 4. handleSummary - custom summary handling
export function handleSummary(data) {
  return {
    'summary.json': JSON.stringify(data),
  };
}
```

Test Structure Best Practices

javascript

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

// Test configuration
export let options = {
  vus: 10,
  duration: '1m',
};

// Test data
const BASE_URL = 'https://api.example.com';

// Helper functions
function authenticate() {
  // Authentication logic
}

// Main test function
export default function () {
  // Test steps
  let response = http.get(`${BASE_URL}/endpoint`);

  // Validations
  check(response, {
    'status is 200': (r) => r.status === 200,
  });

  // Think time
  sleep(1);
}
```

---

6. HTTP Testing {#http-testing}

Basic HTTP Methods

javascript

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

export default function () {
  // GET request
  let getResponse = http.get('https://httpbin.org/get');

  // POST request with JSON data
  let postData = JSON.stringify({
    name: 'John',
    email: 'john@example.com'
  });

  let postResponse = http.post('https://httpbin.org/post', postData, {
    headers: {
      'Content-Type': 'application/json',
    },
  });

  // PUT request
  let putResponse = http.put('https://httpbin.org/put', postData, {
    headers: {
      'Content-Type': 'application/json',
    },
  });

  // DELETE request
  let deleteResponse = http.del('https://httpbin.org/delete');
}
```

Request Parameters and Headers

javascript

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

export default function () {
  // With query parameters
  let params = {
    headers: {
      'Authorization': 'Bearer token123',
      'User-Agent': 'k6-test',
    },
    timeout: '60s',
  };

  let response = http.get('https://httpbin.org/get?param1=value1&param2=value2', params);
}
```

Form Data

javascript

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

export default function () {
  // Form data
  let formData = {
    username: 'testuser',
    password: 'testpass',
  };

  let response = http.post('https://httpbin.org/post', formData);
}
```

File Uploads

javascript

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

export default function () {
  let binFile = open('/path/to/file.jpg', 'b');

  let data = {
    file: http.file(binFile, 'test.jpg', 'image/jpeg'),
    field: 'value',
  };

  let response = http.post('https://httpbin.org/post', data);
}
```

---

7. Checks and Thresholds {#checks-thresholds}

Checks (Assertions)

Checks verify that your application behaves correctly:

javascript

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

export default function () {
  let response = http.get('https://httpbin.org/get');

  check(response, {
    // Status code checks
    'status is 200': (r) => r.status === 200,
    'status is not 404': (r) => r.status !== 404,

    // Response time checks
    'response time < 500ms': (r) => r.timings.duration < 500,
    'response time < 1s': (r) => r.timings.duration < 1000,

    // Content checks
    'response contains expected text': (r) => r.body.includes('httpbin'),
    'response is JSON': (r) => r.headers['Content-Type'] === 'application/json',

    // Header checks
    'has correct content-type': (r) => r.headers['Content-Type'] !== undefined,

    // Body size checks
    'response body size': (r) => r.body.length > 0,
  });
}
```

Thresholds (Pass/Fail Criteria)

Thresholds define when a test should pass or fail:

javascript

```javascript
export let options = {
  vus: 10,
  duration: '1m',
  thresholds: {
    // HTTP request duration thresholds
    'http_req_duration': ['p(95)<500', 'p(99)<1000'],

    // HTTP request failure rate
    'http_req_failed': ['rate<0.1'], // Less than 10% failures

    // Checks success rate
    'checks': ['rate>0.95'], // 95% of checks must pass

    // Custom metric thresholds
    'http_req_duration{status:200}': ['max<2000'],
    'http_req_duration{status:500}': ['max<1000'],

    // Iteration duration
    'iteration_duration': ['avg<2000'],

    // Data transfer
    'data_received': ['count>1000000'], // At least 1MB received
  },
};
```

Advanced Threshold Examples

javascript

```javascript
export let options = {
  thresholds: {
    // Percentile thresholds
    'http_req_duration': [
      'p(50)<200',  // 50% of requests under 200ms
      'p(95)<500',  // 95% of requests under 500ms
      'p(99)<1000', // 99% of requests under 1000ms
    ],

    // Conditional thresholds
    'http_req_duration{expected_response:true}': ['p(95)<300'],
    'http_req_duration{name:login}': ['p(95)<400'],

    // Rate thresholds
    'http_req_failed': ['rate<0.05'], // Less than 5% failure rate
    'checks': ['rate>0.99'],        // 99% check success rate

    // Count thresholds
    'http_reqs': ['count>1000'],     // At least 1000 requests
    'vus': ['value>10'],           // At least 10 VUs
  },
};
```

---

8. Load Testing Scenarios {#scenarios}

Basic Load Test Patterns

Constant Load

javascript

```javascript
export let options = {
  vus: 50,
  duration: '5m',
};
```

Ramping Load

javascript

```javascript
export let options = {
  stages: [
    { duration: '1m', target: 10 }, // Ramp up to 10 VUs over 1 minute
    { duration: '3m', target: 10 }, // Stay at 10 VUs for 3 minutes
    { duration: '1m', target: 0 },  // Ramp down to 0 VUs over 1 minute
  ],
};
```

Stress Testing

javascript

```javascript
export let options = {
  stages: [
    { duration: '2m', target: 50 },  // Normal load
    { duration: '5m', target: 50 },  // Stay at normal load
    { duration: '2m', target: 100 }, // Increase load
    { duration: '5m', target: 100 }, // Stay at increased load
    { duration: '2m', target: 200 }, // Spike to high load
    { duration: '5m', target: 200 }, // Stay at high load
    { duration: '2m', target: 0 },   // Ramp down
  ],
};
```

Advanced Scenarios

Multiple Scenarios

javascript

```javascript
export let options = {
  scenarios: {
    constant_load: {
      executor: 'constant-vus',
      vus: 10,
      duration: '5m',
    },
    spike_test: {
      executor: 'ramping-vus',
      startVUs: 0,
      stages: [
        { duration: '30s', target: 50 },
        { duration: '1m', target: 50 },
        { duration: '3
```