

Topic: "🎯 Goals:

- Understand performance testing
- Install and run K6
- Write your first test"




Tool: K6 for Performance testing

You: Expert in Coding in tool, good guide in explain code step by step from procedure to open a file to Different run command used to run the code, Code having best comments for beginner to understand and its from beginner to advance About a given topic, Covers all the topics related to the given topic in one response without asking any questions which makes me feel satisfied that all the necessary content of topic is present.

Absolutely! Here's a **complete beginner-to-advanced guide** on **K6 Performance Testing** using the tool effectively, with **code**, **step-by-step explanation**, **setup instructions**, and **commands** — all in one place. Let's dive in! 🚀

---

## GOALS

-  Understand Performance Testing
-  Install and Run K6
-  Write Your First Test

-  Learn All Core Concepts from Beginner to Advanced
- 

## WHAT IS PERFORMANCE TESTING?

Performance testing is a type of non-functional testing that determines how an application behaves under various loads. Its goal is to ensure:

- The system's **speed**, **scalability**, **reliability**, and **stability** under workload.
- Detect **bottlenecks** before users do.

**K6** is an open-source, modern load testing tool built with Go and JavaScript for developers and testers.

---

## INSTALLING K6

### ♦ For Windows:

Install [Chocolatey](#) (if not already):

```
Set-ExecutionPolicy Bypass -Scope Process -Force; `
[System.Net.ServicePointManager]::SecurityProtocol = `
[System.Net.ServicePointManager]::SecurityProtocol -bor 3072; `
iex ((New-Object
System.Net.WebClient).DownloadString('https://community.chocolatey.org/install.ps1'))
```

1.

Install K6:

```
choco install k6
```

2.

### ♦ For macOS:

```
brew install k6
```

### ◆ For Linux (Debian/Ubuntu):

```
sudo apt update
sudo apt install gnupg software-properties-common
curl -s https://dl.k6.io/key.gpg | sudo apt-key add -
echo "deb https://dl.k6.io/deb stable main" | sudo tee /etc/apt/sources.list.d/k6.list
sudo apt update
sudo apt install k6
```

✅ To verify installation:

k6 version

---

## WRITE YOUR FIRST TEST

### Step-by-Step: Create and Run a Test File

1. **Open your text editor/IDE (e.g., VS Code)**

Create a new file:

test-script.js

- 2.
3. **Paste the following code:**

```
// test-script.js

import http from 'k6/http';      // Importing HTTP module for making requests
import { check, sleep } from 'k6'; // Importing utility functions

export const options = {
  vus: 10,           // Virtual Users (simulate 10 users)
  duration: '30s',   // Total duration of the test
};

// The default function is the entry point for VUs
export default function () {
  const res = http.get('https://test.k6.io'); // Sending a GET request to the test site
```

```
// Validate response status
check(res, {
  'Status is 200': (r) => r.status === 200,
});

sleep(1); // Simulate user think-time of 1 second
}
```

---

## **RUNNING THE TEST**

Open terminal where `test-script.js` is located and run:

```
k6 run test-script.js
```

You'll see live metrics like:

- **Checks** passed/failed
  - **HTTP requests/sec**
  - **Failures, Duration, HTTP codes**
- 

## **ADVANCED OPTIONS & FEATURES**

### ♦ **1. Thresholds (Pass/Fail Criteria)**

```
export const options = {
  thresholds: {
    http_req_duration: ['p(95)<500'], // 95% of requests must be under 500ms
  },
};
```

### ♦ **2. Ramp-up Stages (Gradual Load)**

```
export const options = {
  stages: [
    { duration: '30s', target: 20 }, // Ramp to 20 users
    { duration: '1m', target: 20 }, // Stay at 20 users
  ]
};
```

```

    { duration: '30s', target: 0 }, // Ramp-down to 0
  ],
};

```

### ♦ 3. Multiple Requests & Custom Headers

```

export default function () {
  const headers = {
    'Content-Type': 'application/json',
  };

  const payload = JSON.stringify({ name: "Routh", role: "Tester" });

  const res = http.post('https://test-api.k6.io/user', payload, { headers });

  check(res, {
    'Created': (r) => r.status === 201,
  });

  sleep(1);
}

```

### ♦ 4. Data-driven testing with CSV

```

import { SharedArray } from 'k6/data';
import http from 'k6/http';

const data = new SharedArray('users', function () {
  return JSON.parse(open('./users.json')); // Or CSV with papaparse
});

export default function () {
  const user = data[__VU % data.length]; // Use data per Virtual User
  http.post('https://example.com/login', JSON.stringify(user));
}

```

---

## USEFUL COMMANDS

Command	Description
<code>k6 run script.js</code>	Run test in CLI

<code>k6 run --vus 20 --duration 1m script.js</code>	Override script options
<code>k6 run --summary-export=result.json script.js</code>	Export result to JSON
<code>k6 archive script.js -o archive.tar</code>	Archive the script
<code>k6 inspect archive.tar</code>	Inspect an archived test
<code>k6 cloud script.js</code>	Run test in K6 Cloud (requires account)

---

## VIEWING RESULTS

K6 by default shows results in the CLI. For more advanced visualization:

- Use **k6 Cloud** (free plan available)
  - Export results to:
    - JSON (`--summary-export`)
    - InfluxDB + Grafana for live dashboards
- 

## MONITORING METRICS

Metric	Description
<code>http_req_duration</code>	Time taken for HTTP request
<code>http_req_failed</code>	Number of failed requests
<code>vus</code>	Number of active virtual users
<code>iterations</code>	Total function executions

---

## ✓ TIPS FOR BEST PRACTICES

- Add meaningful `check()` conditions
  - Use `sleep()` to simulate real user behavior
  - Always start small, then ramp up
  - Use `options` to simulate realistic load patterns
  - Keep test cases modular and reusable
- 

## SUMMARY

You've learned:

- ✓ What performance testing is
  - ✓ How to install and verify K6
  - ✓ How to write and run your first test
  - ✓ Key K6 concepts like `vus`, `duration`, `stages`, `thresholds`
  - ✓ Advanced features like data-driven testing, headers, and scripting
- 

Would you like me to generate a **template project folder** with `README.md`, test files, and a directory structure ready to run?

## 1. What is Performance Testing? ○ Load Testing ○ Stress Testing ○ Soak Testing ○ Spike Testing

Sure! Here's a complete, beginner-to-advanced guide on **Performance Testing** using the **K6** tool, covering the key types: **Load**, **Stress**, **Soak**, and **Spike Testing**—along with well-commented K6 scripts, how to run them, and detailed explanations.

---

## 1. What is Performance Testing?

**Performance Testing** is a type of software testing that ensures a system performs well under expected or unexpected workloads. The goal is to check **speed, scalability, stability, and responsiveness** of an application.

### Four Main Types:

Type	Description
<b>Load Testing</b>	Measures the system's performance under expected user load.
<b>Stress Testing</b>	Determines the upper limit and how the system behaves under extreme conditions.
<b>Soak Testing</b>	Evaluates system stability over a prolonged period under a normal load.
<b>Spike Testing</b>	Tests the system's reaction to sudden large spikes in the load.

---

### Tool Used: K6 by Grafana Labs

K6 is a modern, open-source load testing tool written in Go, with scripts written in JavaScript.

---

## Step-by-Step Guide: From Setup to Running Tests

### Step 1: Install K6

#### For Windows:

```
choco install k6
```

#### For Mac:

```
brew install k6
```

#### For Linux (Debian/Ubuntu):



```
sudo apt install gnupg ca-certificates
curl -fsSL https://dl.k6.io/key.gpg | sudo gpg --dearmor -o
/usr/share/keyrings/k6-archive-keyring.gpg
echo "deb [signed-by=/usr/share/keyrings/k6-archive-keyring.gpg] https://dl.k6.io/deb stable
main" | sudo tee /etc/apt/sources.list.d/k6.list
sudo apt update
sudo apt install k6
```

---

## ✅ Step 2: Create a Test File

```
mkdir k6-performance-tests
cd k6-performance-tests
touch test-script.js
```

---

## ✅ Step 3: Sample K6 Test Script Covering All Test Types

```
// test-script.js

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

// Define test options
export let options = {
  stages: [
    { duration: '30s', target: 10 }, // Load testing - ramp up to 10 users
    { duration: '1m', target: 10 }, // Soak testing - hold at 10 users
    { duration: '10s', target: 50 }, // Spike testing - spike to 50 users
    { duration: '10s', target: 0 }, // Ramp down
  ],
  thresholds: {
    http_req_duration: ['p(95)<500'], // 95% of requests should be below 500ms
  },
};

// Main function simulates a virtual user (VU)
export default function () {
  const response = http.get('https://test.k6.io'); // Replace with your endpoint

  // Check if response is 200 OK
  check(response, {
    'status is 200': (r) => r.status === 200,
  });
}
```

```
});  
  
sleep(1); // Simulate user think time  
}
```

---

## Explanation of Script

- **import**: Load required modules.
  - **options**:
    - **stages**: Control how virtual users ramp up/down.
      - First 30s → ramp up to 10 (Load Test)
      - Then 1m → stay at 10 (Soak Test)
      - Next 10s → spike to 50 (Spike Test)
      - Finally, ramp down to 0 (End test)
    - **thresholds**: Validate performance goals (e.g., 95% responses < 500ms)
  - **default function**: Each Virtual User (VU) will execute this block.
    - Makes a GET request.
    - Verifies response status is 200.
    - Waits 1 second (simulates user think time).
- 

## Additional Test Types in Isolation

### Load Testing (Only):

```
export let options = {  
  vus: 50,      // 50 virtual users  
  duration: '30s', // run for 30 seconds
```

```
};
```

### ▶ **Stress Testing (Push limits):**

```
export let options = {  
  stages: [  
    { duration: '20s', target: 50 },  
    { duration: '20s', target: 100 },  
    { duration: '20s', target: 200 },  
    { duration: '20s', target: 0 },  
  ],  
};
```

### ▶ **Soak Testing (Long-duration stability):**

```
export let options = {  
  vus: 20,  
  duration: '1h',  
};
```

### ▶ **Spike Testing (Sudden surge):**

```
export let options = {  
  stages: [  
    { duration: '10s', target: 0 },  
    { duration: '5s', target: 100 }, // sudden spike  
    { duration: '10s', target: 0 },  
  ],  
};
```

---



## **How to Run the Test**

### ▶ **Basic Command:**

```
k6 run test-script.js
```

### ▶ **With Output in Summary:**

```
k6 run --summary-trend-stats="avg,p(95),min,max" test-script.js
```

## Save Output to JSON:

`k6 run --out json=result.json test-script.js`

---



## View Performance Results

K6 shows:

- HTTP response codes
  - Request duration (avg, p95, max, etc.)
  - Number of iterations
  - VUs behavior over time
  - Any failed requests
- 







## Reusability Tips

- Modularize endpoints in constants
  - Use `setup()` and `teardown()` functions for global data
  - Use `groups()` to logically separate API tests
- 



## Summary

Topic	Covered 
What is Performance Testing	
Load Testing with K6	
Stress Testing with K6	

Soak Testing with K6	✓
Spike Testing with K6	✓
Installation Steps	✓
Full Example with Comments	✓
Commands to Run & View Data	✓

---

If you follow the above guide, you'll go from a **beginner to confident performance tester using K6**. Let me know when you're ready to scale to **distributed testing**, **CI/CD integrations**, or **Grafana dashboards** for real-time analysis!

## K6 Introduction ○ What is K6 and why use it? ○ K6 vs JMeter vs Gatling

Sure! Here's a **complete and beginner-to-advanced guide** on the topic "**K6 Introduction – What is K6 and why use it? | K6 vs JMeter vs Gatling**", with **step-by-step instructions**, **clearly explained code examples**, and **commands to run them**. This guide will help you understand **performance testing using K6** thoroughly.

---

## K6 Introduction

### ✓ What is K6?






**K6** is an **open-source, modern performance testing tool** written in **Go (backend)** with test scripts written in **JavaScript (frontend)**. It is used to test the **performance**, **reliability**, and **scalability** of web applications and APIs.

- Developed by **Grafana Labs**
- Designed for **developer-centric load testing**
- Supports **automated CI/CD workflows**

- Lightweight, high-performance, and very **scriptable**

---

## Why Use K6?

Feature	Benefit
 JavaScript-based scripting	Easy to write and maintain test cases
 CLI-based execution	No GUI needed, runs fast on any machine
 CI/CD friendly	Can be integrated in Jenkins, GitHub Actions, GitLab CI, etc.
 Metrics	Provides rich metrics: latency, errors, throughput, and thresholds
 Scalable	Can run in distributed or cloud mode using k6 Cloud

---

## K6 vs JMeter vs Gatling

Feature	K6	JMeter	Gatling
Language	JavaScript	Java	Scala
Performance	High (written in Go)	Medium (Java overhead)	High
UI	CLI	GUI and CLI	CLI
Scripting Ease	Easy (JS-based)	XML-heavy (unless plugins used)	Moderate (Scala knowledge needed)
CI/CD Ready	Yes	Yes (with plugins)	Yes
Community	Growing fast	Mature	Moderate
Reporting	Built-in + external tools	Rich built-in GUI reports	Excellent HTML reports

---



# Getting Started with K6



## Installation

### On Windows:

```
choco install k6
```

### On macOS:

```
brew install k6
```

### On Linux:

```
sudo apt install gnupg ca-certificates
curl -s https://dl.k6.io/key.gpg | sudo apt-key add -
echo "deb https://dl.k6.io/deb stable main" | sudo tee /etc/apt/sources.list.d/k6.list
sudo apt update
sudo apt install k6
```

---



## Step-by-Step Code Example

### Step 1: Create a JS file for your script

```
mkdir k6-tests
cd k6-tests
touch script.js
```

### Step 2: Sample script (**script.js**)

```
// script.js

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

// Options block for configuration
export const options = {
  vus: 10, // Number of Virtual Users
  duration: '30s', // Duration of the test
```

```
};

// Default function that runs for each VU
export default function () {
  // Sending GET request to example.com
  const res = http.get('https://test-api.k6.io/public/crocodiles/');

  // Check for 200 OK response
  check(res, {
    'status is 200': (r) => r.status === 200,
  });

  // Wait for 1 second before next iteration
  sleep(1);
}
```

---

## Running K6 Script

### Basic Run Command

```
k6 run script.js
```

### Run with Different VUs and Duration

```
k6 run --vus 50 --duration 1m script.js
```

---

## Output You'll See

```
running (1m00.00s), 00/50 VUs, 1000 complete and 0 interrupted iterations
default ✓ [=====] 50 VUs 1m0s
```

```
✓ status is 200
```

```
checks.....: 100.00% ✓ 1000 ✗ 0
data_received.....: 200 kB 3.3 kB/s
http_req_duration.....: avg=350ms min=100ms max=900ms
```

---





## Performance Metrics Explained

Metric	Meaning
<code>vus</code>	Virtual users running concurrently
<code>duration</code>	Total time the script will run
<code>http_req_duration</code>	Time taken to get a response from request
<code>check</code>	Assert success/failure of conditions
<code>data_received</code>	Total response data received from server
<code>iterations</code>	Total test iterations completed

---



## Advanced Topics

### 1. Parameterization using Data

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

const users = JSON.parse(open('./users.json'));

export default function () {
  const user = users[Math.floor(Math.random() * users.length)];
  http.post('https://example.com/api/login', JSON.stringify(user), {
    headers: { 'Content-Type': 'application/json' },
  });
  sleep(1);
}
```

`users.json`:

```
[
  { "username": "user1", "password": "pass1" },
  { "username": "user2", "password": "pass2" }
]
```

Run:

```
k6 run script.js
```

---

## 2. Thresholds & Performance Budgets

```
export const options = {
  thresholds: {
    http_req_duration: ['p(95)<500'], // 95% of requests must be < 500ms
  },
};
```

---

## 3. Stages for Ramp-Up Testing

```
export const options = {
  stages: [
    { duration: '1m', target: 20 },
    { duration: '2m', target: 40 },
    { duration: '2m', target: 0 },
  ],
};
```

---

## 4. Tags & Groups

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

export default function () {
  group('Homepage Load', function () {
    http.get('https://test.k6.io');
  });

  group('Login API', function () {
    http.post('https://test.k6.io/login', {
      email: 'test@example.com',
      password: '123456',
    });
  });
}
```

---








## Optional: K6 Cloud (Optional)

For advanced visualizations:

k6 login cloud  
k6 cloud script.js

---

## Summary

Concept	Covered
What is K6?	
Why use K6?	
K6 vs Others	
Script writing	
Running scripts	
Metrics & checks	
Advanced usage	

---

If you follow this step-by-step, you can go from **K6 beginner to advanced** without confusion. Let me know if you need a **project structure**, **integration with GitHub Actions**, or **reporting dashboards** later on!

## Installation & Setup [Install K6 \(Windows/Linux/Mac\)](#) [Install via Docker or Homebrew](#)

Absolutely! Here's a **complete and beginner-friendly guide** to the topic “**Installation & Setup of K6**” covering:

- Installing K6 on **Windows**, **Linux**, and **Mac**
  - Installing K6 via **Docker** and **Homebrew**
  - Running your **first K6 test script** with **step-by-step commands**
  - Includes **well-commented sample code** and **run commands**
  - Covers **all essential details from beginner to advanced**
- 

## What is K6?

**K6** is an open-source **performance testing tool** for testing the **load**, **stress**, and **scalability** of your applications, APIs, or websites using JavaScript-based scripting.

---

## 1. Install K6 - OS-wise Guide

### Windows Installation

#### A. Using Chocolatey (Recommended)

Chocolatey is a package manager for Windows

1. Open **PowerShell as Administrator**

Install Chocolatey (if not already):

```
Set-ExecutionPolicy Bypass -Scope Process -Force; `
[System.Net.ServicePointManager]::SecurityProtocol = `
[System.Net.ServicePointManager]::SecurityProtocol -bor 3072; `
iex ((New-Object System.Net.WebClient).DownloadString('https://chocolatey.org/install.ps1'))
```

- 2.

Install K6:

```
choco install k6
```

3.

Verify:

k6 version

4.

---

## ✅ Linux Installation

### A. Using Official Debian/Ubuntu Packages:

```
sudo apt update
sudo apt install gnupg ca-certificates
curl -fsSL https://dl.k6.io/key.gpg | sudo gpg --dearmor -o
/usr/share/keyrings/k6-archive-keyring.gpg
echo "deb [signed-by=/usr/share/keyrings/k6-archive-keyring.gpg] https://dl.k6.io/deb stable
main" | sudo tee /etc/apt/sources.list.d/k6.list
sudo apt update
sudo apt install k6
```

### B. Verify:

k6 version

---

## ✅ Mac Installation

### A. Using Homebrew (Recommended)

```
brew install k6
```

### B. Verify:

k6 version

---



## 2. Install K6 via Docker

- ◆ **Pre-requisite: Docker must be installed**

Check:

```
docker --version
```

### ♦ **Run K6 via Docker:**

You can run K6 without installing it natively:

```
docker run -i grafana/k6 run - <script.js
```

♦ This command:

- Pulls the K6 image from Docker Hub
- Takes input JavaScript from stdin (pipe a local script)

---

## **3. File Setup for First K6 Test**

### ♦ **Create a test file**

Create a file named `script.js` with the following content:

```
// script.js

import http from 'k6/http'; // Import the HTTP module from K6
import { check, sleep } from 'k6'; // Import check and sleep functions

// Options: define number of virtual users and test duration
export const options = {
  vus: 10, // Number of virtual users
  duration: '30s', // Total test duration
};

// Default function where each VU runs this logic
export default function () {
  // Send a GET request to a sample website
  let response = http.get('https://test.k6.io');

  // Check the response status
  check(response, {
    'status is 200': (r) => r.status === 200,
```

```
});  
  
// Wait for 1 second before next iteration  
sleep(1);  
}
```

---

## 4. Running the Test

### A. Run the Script Locally (OS installation)

```
k6 run script.js
```

### B. Run the Script via Docker

```
docker run -i -v $PWD:/scripts grafana/k6 run /scripts/script.js
```

On Windows with PowerShell:

```
docker run -i -v ${PWD}:/scripts grafana/k6 run /scripts/script.js
```

---

## 5. Output Summary (After Running)

You'll see:

- Request counts
  - Status codes
  - Response times
  - Data transfer stats
  - Pass/fail checks
- 

## BONUS: Advanced Run Options

## Change VUs & Duration via CLI (Overrides script)

```
k6 run --vus 50 --duration 1m script.js
```

## Run with Ramp-up Pattern

```
// ramping-test.js
```

```
export const options = {
  stages: [
    { duration: '30s', target: 20 }, // ramp up
    { duration: '1m', target: 20 }, // hold
    { duration: '30s', target: 0 }, // ramp down
  ],
};
```

```
export default function () {
  http.get('https://test.k6.io');
  sleep(1);
}
```

Run:

```
k6 run ramping-test.js
```

---

## Summary of Commands

Action	Command
Verify K6 installed	<code>k6 version</code>
Run basic test	<code>k6 run script.js</code>
Run with Docker	<code>docker run -i -v \$PWD:/scripts grafana/k6 run /scripts/script.js</code>
Custom VUs & duration	<code>k6 run --vus 10 --duration 10s script.js</code>
View help	<code>k6 help</code>

---



## ✓ Recap: Topics Covered

- ✓ Installing K6 on **Windows/Linux/Mac**
  - ✓ Installing via **Chocolatey / Homebrew / Docker**
  - ✓ Creating and saving test file ( **.js** )
  - ✓ Running tests with basic and advanced options
  - ✓ Step-by-step CLI commands
  - ✓ Well-commented example for beginners
  - ✓ Advanced setup with **stages**
- 

Let me know whenever you're ready for the next topic: **Scripting Essentials in K6**, **Thresholds**, or **Integrating with Grafana Cloud/InfluxDB**.

Want this guide as a downloadable **.pdf** or **.md**? Just say the word!

First Test ○ Writing a simple script ○ Running the test using k6 run script.js Basic Script Structure

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

Absolutely! Here's a **complete beginner-to-advanced explanation and guide** for the topic **“First Test in K6”**, including how to create, write, run, and understand a **simple K6 performance test script**. This includes detailed comments, steps to run the script, and extra advanced tips to go beyond the basics.

---

## ✓ Topic: First Test using K6 (Performance Testing Tool)

---

### 1. What is K6?

K6 is an open-source modern load testing tool built for testing the performance of APIs, microservices, and websites. It uses JavaScript to define test cases, making it easy for developers and testers.

---

### 2. Prerequisites

- Node.js installed (for running JS-related tools)
- K6 installed

#### Install K6:

##### Windows (using Chocolatey):

```
choco install k6
```

- 

##### Mac (using Homebrew):

```
brew install k6
```

- 

##### Linux (via curl):

```
sudo apt install gnupg ca-certificates
curl -s https://dl.k6.io/key.gpg | sudo gpg --dearmor -o
/usr/share/keyrings/k6-archive-keyring.gpg
echo "deb [signed-by=/usr/share/keyrings/k6-archive-keyring.gpg] https://dl.k6.io/deb stable
main" | sudo tee /etc/apt/sources.list.d/k6.list
sudo apt update
sudo apt install k6
```

-

---

## 3. Create Your First Test File

### Step-by-step to Create File

1. Open terminal or command prompt.

Create a folder (optional but cleaner):

```
mkdir k6-tests && cd k6-tests
```

- 2.

Create a file:

```
touch script.js
```

- 3.

Open the file in your editor (VSCode recommended):

```
code script.js
```

- 4.
- 

## 4. Write the First Script

```
// script.js
```

```
// Importing required modules from K6
```

```
import http from 'k6/http'; // For making HTTP requests
```

```
import { check, sleep } from 'k6'; // check is for validations, sleep pauses test for a while
```

```
// Default function that runs during the test
```

```
export default function () {
```

```
  // Send a GET request to the target URL
```

```
  let res = http.get('https://test.k6.io');
```

```
  // Perform a check to validate if status is 200
```

```
  check(res, {
```

```
    'status is 200': (r) => r.status === 200, // Check that HTTP response status is 200 OK
```

```
});  
  
// Sleep for 1 second to simulate realistic user think time  
sleep(1);  
}
```

---

## 5. Run the Test

### Basic Run Command

k6 run script.js

### Output:

You'll see details like:

- HTTP requests per second
  - Number of VUs (Virtual Users)
  - Duration
  - Pass/Fail results of your checks
- 

## 6. Understand the Script (Line-by-Line)

Line	Explanation
<code>import http from 'k6/http';</code>	Loads HTTP module for sending requests.
<code>import { check, sleep } from 'k6';</code>	<code>check()</code> is used for asserting expected behavior; <code>sleep()</code> simulates wait time.
<code>export default function ()</code>	The main test function K6 runs repeatedly.
<code>http.get('https://test.k6.io');</code>	Sends GET request to sample website.

```
check(res, {...});
```

Validates that the response meets expectations (e.g., status 200).

```
sleep(1);
```

Simulates user think-time (1 second).

---

## 7. Advanced Usage (Additions for Real-World Testing)

### Add Virtual Users (VUs) and Duration:

```
k6 run --vus 10 --duration 30s script.js
```

- `--vus`: Number of Virtual Users
  - `--duration`: Total test duration
- 

### Save Output Report:

```
k6 run --out json=results.json script.js
```

### Run with HTML Report (using external tools like `k6-reporter`):

1. Install `k6-reporter` (Node-based tool).
  2. Convert JSON to HTML.
- 

## 8. Bonus Tips

Tip	Description
<code>http.post( )</code>	Send POST requests (useful for API testing).
<code>http.batch( )</code>	Send multiple requests in parallel.

`options`      Send headers, payload, etc.  
object

`setup()`      Prepare data before tests run.

`teardown()`      Cleanup actions post test run.

---

## 9. Script Template for Reuse







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

export let options = {
  vos: 5, // Number of virtual users
  duration: '10s', // Duration of the test
};

export default function () {
  let res = http.get('https://test.k6.io');
  check(res, {
    'response code was 200': (r) => r.status === 200,
    'body size is > 0': (r) => r.body.length > 0,
  });
  sleep(1);
}
```

---

## 10. Summary Checklist

- |  | Task   |
|---|--|
|  | Installed K6                                       |
|  | Created a test script file                         |
|  | Wrote a basic test with <code>http.get</code>      |
|  | Added <code>check()</code> for validation          |
|  | Ran the script using <code>k6 run script.js</code> |

- ✓ Learned to add VUs and duration
  - ✓ Explored basic customization (sleep, options)
  - ✓ Bonus: Learned batch requests and reporting
- 

If you follow all the steps above, **you'll have a solid understanding of how to write and execute your first performance test using K6**, from start to advanced basics — without missing anything.

Let me know when you're ready for the next step like **batch testing**, **parameterization**, **authentication headers**, or **HTML reports**.