Load testing examples

Related content
Distributed load tests on Kubernetes 12 min read
Load testing with Postman 10 min read
Load test OAuth APIs 9 min read
Load test SQL databases 10 min read

Load testing is a type of software testing that helps ensure an application can handle expected workloads, while maintaining stable performance and providing a good end-user experience. While teams can have multiple goals when conducting a load test, the primary objective is to simulate the average amount of activity on a system during a typical day in production.

Here, we'll take a look at some common steps involved in the load testing process, as well as walk through some specific load testing examples.

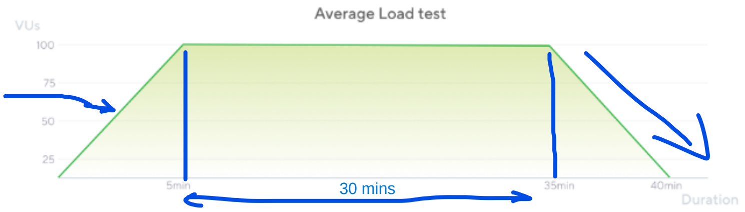
How do you perform load testing?

How do you perform the load testing?
- Ramp-up period: Increasing the scripts activity until it reaches the desired number of simulated user and throughput
- Plateau period: Maintaining the desired load for a set period of time.
- Ramp-down period: Letting it ramp down gradually

The exact process you follow to perform a load test can vary, based on the system or application you want to test, as well as your overall goals and requirements. That said, to perform a simple load test, you will likely use a load testing tool, such as Grafana k6, to create and run a test script.

Then, the test itself will generally entail the follow:

1. Increasing the script's activity until it reaches the desired number of simulated users and throughput. This is sometimes known as a ramp-up period, and typically lasts between 5% and 15% of the total test duration. This ramp-up period gives your system time to warm up or auto-scale to handle the traffic.
2. Maintaining your desired load for a set period of time. A general best practice here is to aim for an average duration at least five times longer than the ramp-up period, so you can assess performance over a significant window of time.
3. Depending on the test case, either stopping the test completely after that set period of time or letting it ramp down gradually.



Ram up time = 5 min, 12.5% of total test duration

In an average load test, you gradually ramp up to the desired value of virtual users or throughput, stay there for the indicated period, then ramp down.

What are the steps in load testing?

Again, the exact steps you follow during the load testing process can vary, but testers generally progress through a series of steps, including:

What are the steps in load testing?
Step 1: Defining testing requirements and scope
Step 2: Creating test scripts
Step 3: Running the scripts
Step 4: Analyzing test results

1. **Defining testing requirements and scope**: This entails choosing the specific features, functions, or user journeys you want to test. For whichever system you are testing, be sure to know the specific number of users and the typical throughput per process. To find this, look through application performance monitoring (APM) or analytic tools that provide information from the production environment. If you can't find this information, estimate these numbers as best as you can.
2. **Creating the test script**: Using a load testing tool like Grafana k6, you will build a script that automates the load test.
3. **Running the script**: The script will progress until it reaches the defined number of users and throughput. Then, that load will be maintained for a specified period, before the test either stops or gradually ramps down.
4. **Analyzing test results**: Review the generated test results to understand whether the system's performance and resource consumption remained stable during the period of full load. In some cases, a system may perform poorly during this period – a sign that there may be an underlying issue you need to identify and address. On the other hand, even if a system performs well under the load, you might want to perform further tests, such as a stress test, to assess how it performs under above-average load.

What are examples of load testing?

Load testing benefits any application and encompasses a broad spectrum of use cases, but here are some common examples of applications or systems you might load test:

What are examples of load testing?
- Ecommerce websites
- Banking applications
- Healthcare applications
- Video streaming platforms

Steps of testing, all are http requests sending, latency checking, iteration checking and more

1. **Ecommerce website**. In this scenario, the tester simulates users browsing products, adding items to carts, and completing purchases. They also account for different types of users, such as new and returning customers, as well as different types of browsing behavior, such as searching for specific products or browsing categories.
2. **Banking application**. The tester simulates users performing different types of transactions, such as fund transfers, balance inquiries, and load applications. They include different types of users, such as personal and business customers, and different types of transaction volumes

different types of transaction volumes.

3. **Healthcare application**. Virtual users — representing patients or health care providers — access medical records, schedule appointments, and request prescriptions. The test should address different types of usage patterns, such as peak appointment scheduling times.
4. **Video streaming platform**. When load testing a video streaming platform, teams simulate droves of users watching videos, searching for content, and interacting with the platform's features. Testing can also simulate different viewing behaviors, including streaming on different devices and at different resolutions.

In addition to these types of websites and applications, teams might load test other system components, such as an API.

How to do load testing online

In general, the goal of load testing a website is to simulate real user traffic to ensure reliability and prevent failures. You can choose from various website testing approaches, including:

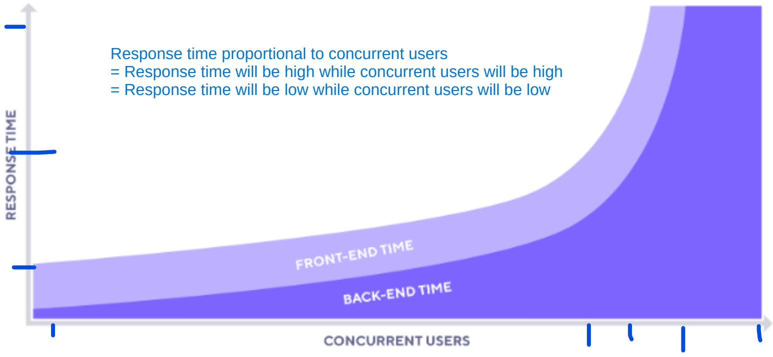
How to do load testing online
- Backend vs. Frontend performance testing
- Protocol-based, browser-based or hybrid load testing
- Component testing vs. end-to-end testing

- **Backend vs. frontend performance testing**: Backend performance testing targets the servers that host the site, while frontend testing focuses more on the end-user experience at the interface level.
- **Protocol-based, browser-based, or hybrid load testing**: Protocol-based load testing simulates the requests that support user actions, such as HTTP requests, while browser-based testing simulates how users access your website via a browser. Hybrid load testing combines these two approaches.
- **Component testing vs. end-to-end testing**: Component testing focuses on the performance of a specific website component or feature, while end-to-end testing tracks the effects of user behavior across the entire stack.

Backend testing: Server site performance / load testing
Frontend testing; end-user experience at the interface level

Protocol based testing: HTTP request
Browser based testing: User access the browser

Component testing: Individual module / component / feature of any website
E2E testing: User behavior across the entire stack



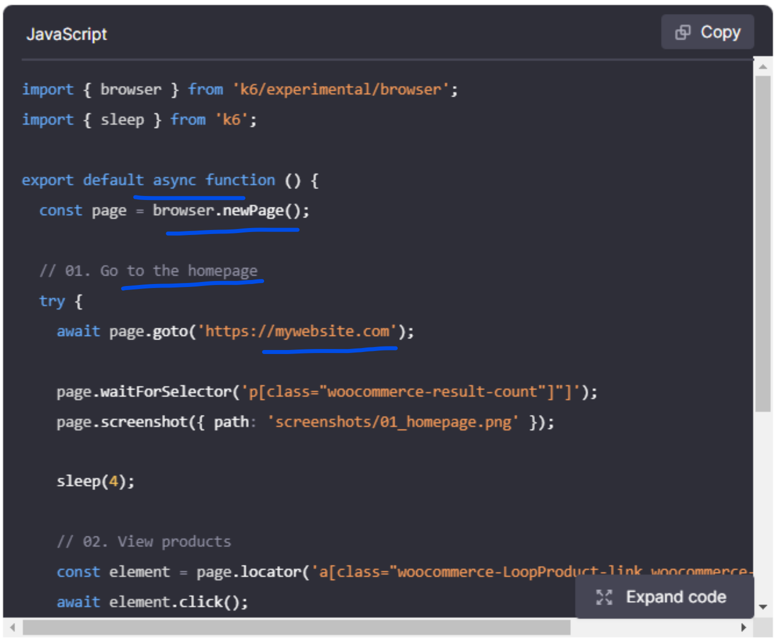
Response time proportional to concurrent users
= Response time will be high while concurrent users will be high
= Response time will be low while concurrent users will be low

Backend performance can significantly affect the overall user experience.

To illustrate what the website load testing process might look like, let's walk through an example of browser-based load testing. Again, the goal of browser-based testing is to assess frontend performance by simulating real users accessing a website via a browser.

A browser-based load testing script, for instance, might include instructions to navigate to a web page, click on a button, and enter text into a form. Those user actions then trigger underlying requests on the protocol layer.

The following is an example of a browser-based load testing script in Grafana k6 — specifically, using the k6 browser module. The script accesses a homepage then simulates a user searching for and clicking on a link to a product page:



```javascript
import { browser } from 'k6/experimental/browser';
import { sleep } from 'k6';

export default async function () {
  const page = browser.newPage();

  // 01. Go to the homepage
  try {
    await page.goto('https://mywebsite.com');

    page.waitForSelector('p[class="woocommerce-result-count"]');
    page.screenshot({ path: 'screenshots/01_homepage.png' });

    sleep(4);

    // 02. View products
    const element = page.locator('a[class="woocommerce-LoopProduct-link woocommerce-
    await element.click();
```

Load testing and stress testing example

As mentioned earlier, load testing is a type of software testing that puts a simulated workload on a system — application, API, or website — to see how it performs. Stress testing is a specific type of load testing that assesses how the system performs when the workload is heavier than usual.

The steps in the stress testing process are generally similar to the steps outlined earlier for loading testing — they just involve a higher load. Below is an example of a stress testing script in Grafana k6 that ramps up to more than 200 simulated users and maintains that volume for 30 minutes:

JavaScript Copy

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

export const options = {
  // Key configurations for Stress in this section
  stages: [
    { duration: '10m', target: 200 }, // traffic ramp-up from 1 to a higher 200 user
    { duration: '30m', target: 200 }, // stay at higher 200 users for 30 minutes
    { duration: '5m', target: 0 }, // ramp-down to 0 users
  ],
};

export default () => {
  const urlRes = http.get('https://test-api.k6.io');
  sleep(1);
  // MORE STEPS
  // Here you can have more steps or complex script
  // Step1
```

Expand code

k6 load testing example

Grafana k6 is an open source load testing tool. Teams can use k6 to test system reliability and performance, and more quickly identify issues. Grafana Cloud k6 is the hosted and fully managed version of Grafana k6.

Some of k6's biggest advantages are its code-based scripting and the fact that it caters to the developer experience.

k6 is structured around four main pillars:

1. **Enabling users to script and configure their workloads**: While k6 is written in Go, k6 users outline their workloads using JavaScript, which k6 runs using its goja interpreter. k6 also supports various open source technologies, tools, and protocols, and allows for customization and flexibility via its extensions.
2. **Planning and executing tests**: Users define specific execution scenarios they want to replicate through configurable options. Then, k6 creates an execution plan and carries it out to align with the user's requirements.
3. **Collecting measurements of software performance, such as response time**: k6 collects measurements, and then classifies and aggregates them into metrics, such as response time.
4. **Forwarding results to users**: An end-of-test summary provides users with immediate and actionable insights.

goja interpreter

Now, let's walk through a brief and simple k6 load testing example (*Note: this example assumes you already have k6 installed*).

Let's say your team just created a new login endpoint, but before releasing it, want to test that it's functional. In k6, you could write a test to send a POST request to the new endpoint and create a check for the response status.

First, you need to add logic for the endpoint. To do that, you need to make an HTTP request:

- Import the HTTP module.
- Create a payload to authenticate the user.
- Use the `http.post` method to send your request with the payload to an endpoint.

To test, copy this file and save it as `api-test.js`:

JavaScript Copy

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

export default function () {
  // define URL and payload
  const url = 'https://test-api.k6.io/auth/basic/login/';
  const payload = JSON.stringify({
    username: 'test_case',
    password: '1234',
  });

  const params = {
    headers: {
      'Content-Type': 'application/json',
    },
  };

  // send a post request and save response as a variable
  const res = http.post(url, payload, params);
```

Expand code

Run the script using the `k6 run` command:

Copy

```
k6 run api-test.js
```

After the test finishes, k6 reports the default result summary.

Copy

```
          /\      |‾‾| /‾‾/   /‾‾/
     /\  /  \     |  |/  /   /  /
    /  \/    \    |     (   /   ‾‾\
   /          \   |  |\  \ |  (‾)  |
  / __________ \  |__| \__\ \_____/ .io

  execution: local
     script: api-test.js
     output: -
  ...
```

As an optional step, you can log the response body to the console to make sure you're getting the right response.

JavaScript

Copy

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

  const res = http.post(url, payload, params);

  // Log the request body
  console.log(res.body);
}
```

Add response checks

Once you verify the request is well-formed, add a check that validates whether the system responds with the expected status code.

Update your script so it has the following check function:

JavaScript

Copy

```
// Import necessary modules
import { check } from 'k6';
import http from 'k6/http';

export default function () {
  // define URL and request body
  const url = 'https://test-api.k6.io/auth/basic/login/';
  const payload = JSON.stringify({
    username: 'test_case',
    password: '1234',
  });
  const params = {
    headers: {
      'Content-Type': 'application/json',
    },
  };

  // send a post request and save response as a variable
  const res = http.post(url, payload, params);
```

Expand code

Run the script again:

Copy

```
k6 run api-test.js
```

Inspect the result output for your check. It should look like this:

```
JavaScript                                                    Copy
✓ response code was 200
```

JMeter load testing example

Like Grafana k6, JMeter is an open source load testing tool. It was built entirely in Java by the Apache Foundation. Like Grafana k6, JMeter supports a variety of protocols and provides a detailed summary of test results that helps users analyze system performance and determine next steps.

One of the primary differences between Grafana k6 and JMeter is that, while JMeter users can extend testing scripts using code, the majority of scripting is done via a GUI. One of the biggest benefits of the GUI model is that there's a low barrier to entry; a load testing tool with a GUI can be easier to learn for testers that are used to mostly no-code UIs, such as those in Postman or SoapUI. That said, one potential drawback of the GUI model is that it can add significantly more resource overhead to an application.

Here's a simple example that illustrates how you would test an expected 404 response in JMeter vs. k6. In JMeter, you would click through a GUI and fill in value entry fields. You'd create a new Response Assertion under the test. In the "Response Field to Test" section of the assertion, you'd check the box for "Ignore Status."

Then, you'd add other assertions as you'd like, such as setting the radio in "Response Field to Test" to "Response Code" and setting the "Patterns to Test" to "404."

In k6, you would use the following script code:

```
let response = http.get("http://some.url/");                  Copy
check(res, {
"Status is 404": (r) => r.status === 404
});
```

In general, JMeter is well-suited for traditional software testing teams or those who prefer a GUI-driven testing tool. Meanwhile, Grafana k6 is designed for cross-functional engineer teams, as well as teams who want to integrate load testing into DevOps workflows or CI/CD pipelines.

More load testing examples

Load testing is an important type of software testing to ensure your systems can handle expected workloads while maintaining a high-quality user experience. Before getting started, it can be helpful to review some common load testing examples.

We have resources to guide you through:

- load testing SQL databases
- load testing using GitHub Actions
- load testing with GitLab

If you want to explore more use cases for load testing, check out more examples in our Grafana k6 documentation.

> **An easier way to get started**
> Grafana Cloud is the easiest way to get started with metrics, logs, traces, and dashboards. We have a generous forever-free tier and plans for every use case.
> Sign up for a free Grafana Cloud account →
> Read more about Grafana Cloud →

Grafana Labs

Grafana

Overview
Deployment options
Plugins
Dashboards

Products

Grafana Cloud
Grafana Cloud Status
Grafana Enterprise Stack
Grafana Cloud Application Observability
Grafana Cloud Frontend Observability
Grafana Cloud IRM

Open Source

Grafana
Grafana Loki
Grafana Mimir
Grafana OnCall
Grafana Tempo
Grafana Agent
Grafana Alloy

Learn

Grafana Labs blog
Documentation
Downloads
Community
Community forums
Community Slack
Grafana Champions

Company

The team
Press
Careers
Events
Partnerships
Contact
Getting help

Grafana Cloud k6
Grafana Cloud Logs
Grafana Cloud Metrics
Grafana Cloud Profiles
Grafana Cloud Synthetic Monitoring
Grafana SLO

Grafana k6
Prometheus
Grafana Faro
Grafana Pyroscope
Grafana Beyla
OpenTelemetry
Grafana Tanka
Graphite
GitHub

Community organizers
Grafana ObservabilityCON
GrafanaCON 2024
The Golden Grot Awards
Successes
Workshops
Videos
OSS vs Cloud
Load testing

Merch

Grafana Cloud Status

Sitemap Legal and Security Terms of Service Privacy Policy Trademark Policy