



Load Testing 101: Essential Tips for Testers and Developers





Content

- 3 | Preface
- 4 | Load Testing Basics
- 6 | Goals of Load Testing
- 8 | Preparing Your Load Tests
- 13 | Making It Real: Emulating Real Life Conditions
In Your Load Tests
- 16 | The Dos and Don'ts of Load Testing
- 18 | About LoadNinja

By the time any software development project nears completion, it likely will have gone through numerous tests, particularly in an Agile environment where testing and development happen concurrently. But no matter how many tests you've run, once your application is nearly complete, there's really only one way to know whether or not your software can handle the actual demands your army of end users will soon be placing on it — load testing.

Load testing is the process of putting simulated demand on software, an application, or website in a way that demonstrates its behavior under various conditions. Over the last decade, the importance of load testing has skyrocketed. What was once a simple pre-deployment exercise to ensure a web application could handle the load of multiple users is now an integral part of software development and continuous performance improvement. As websites and web applications become more innovative and complex, load testing poses a significant challenge for teams to 1. Execute properly and 2. Cover all the bases. Though load testing as

a practice has been around for years, we still see well known mobile and web applications getting overwhelmed during peak traffic hours.

As a result, load testing needs to be done more frequently, more effectively, and more efficiently. It's also created a need to simply train more people in the basics of load testing. Given that our lives increasingly rest on software functioning properly — whether it's in medical devices, transportation, communications, defense, or entertainment — software performance has never been more important.

That's what this eBook is ultimately about — a “Load Testing 101” manual to get the new & aspiring testers started with load testing.

We'll cover:

- | Load testing basics
- | How to prepare for load testing
- | Emulating real life conditions in your load tests
- | Load testing dos and don'ts



Load Testing Basics

Load testing is a type of performance testing. Performance testing is a series of testing methods employed to understand how a system performs in terms of responsiveness and stability under a specific set of strains. Oftentimes, teams begin performance testing in conjunction with or after functional testing within the test phase of the software development lifecycle. Each type of performance testing method helps answer a specific question about your application's behavior that helps carve a path for issue identification & iterative performance improvement.

For instance, load testing helps you answer, how will my application behave in production with the typical traffic that we see? Load testing is used to verify your application's behavior under normal and

peak load conditions. This allows you to verify that your application can meet the desired performance objectives; which are often specified in a service level agreement (SLA). Load testing also enables you to measure response times, throughput rates, resource utilization levels, and to identify your application's breaking point, assuming that breaking point occurs below the peak load condition. Load testing helps you check your web server's performance under a massive load, determine its robustness, and estimate its scalability.

Stress testing enables you to evaluate your application's behavior when it is pushed beyond the normal or peak load conditions, and helps you understand 'what is my application's breaking point?' The purpose of web server stress testing is to find



Load Testing



Stress Testing



Capacity Testing



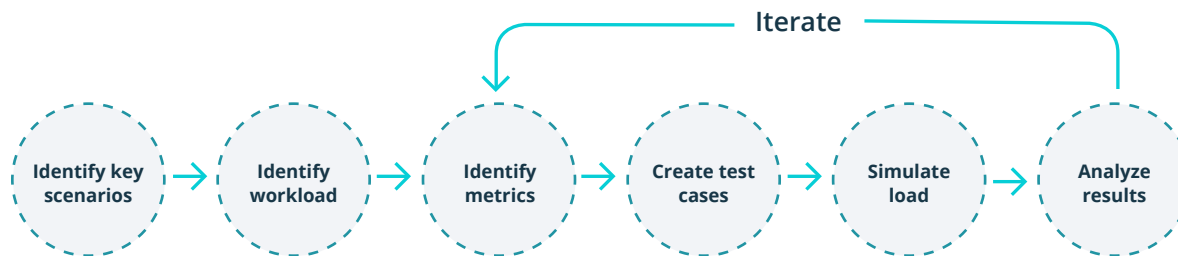
the target application's crash point. The crash point is not always an error message or access violation. It can be a perceptible slowdown in the request processing. The goal of stress testing is to unearth application bugs that surface only under high load conditions. Stress testing helps you find your application's weak points, and how it behaves under extreme load conditions.

Capacity testing is complementary to load testing and determines your server's ultimate failure point, whereas load testing monitors results at various levels of load and traffic patterns. For example, let's say you're developing a new online voting platform,

and you'd like it to be able to handle potentially up to 10,000 user submissions per minute during peak load times. While developing the software, you may have performed unit tests as the code was being written, plus periodic regression tests to make sure you didn't break existing functionality with each new modification as development progressed, but at what point did you begin testing for multiple users?

At what point did you begin testing the program to accept hundreds or even thousands of overlapping field entries, form submissions, and other commands? This is when load testing comes into the picture.

Load testing usually involves the following steps:



Goals of Load Testing

The main goals of load testing are to:

1. Identify bottlenecks and their causes
2. Optimize the application configuration (both the hardware and software) for maximum performance.
3. Verify the reliability of your application under stress

Load/stress testing helps you identify the following characteristics:

- | Response time
- | Throughput
- | Maximum concurrent users supported
- | Resource utilization in terms of the amount of CPU, RAM, network I/O, and disk I/O resources your application consumes during the test
- | Behavior under various workload patterns including normal load conditions, excessive load conditions, and conditions in between
- | Application breaking point
- | Symptoms and causes of application failure under stress conditions
- | Weak points in your application
- | What is required to support a projected increase in load





This eBook keeps the abovementioned steps in mind and uses our load testing tool LoadNinja to help you understand basics of load testing.

We'll approach load testing with a four-step process:

- | **Prepare:** to prepare your web application for load testing
- | **Record:** How to record load testing scenarios
- | **Test:** Creating tests that match real-life circumstances
- | **Analyze:** Understanding and using load testing data



Preparing Your Load Tests

With shift-left and DevOps motion, software teams are pressured to test more and test often. Testing should be performed at each step in the development cycle and should continue after the application is live. While it can be frustrating that a tester's job is never done, it's important to take into consideration that with each testing and remediation cycle, the application improves.

Applications go down under load for two reasons: either developers didn't load test or, worse, they took the time to load test but didn't prepare properly. Without adequate prep work, a load test can't find all the issues that it should.

So, how do you best prepare for a load test? Here are our 10 steps to help you prepare:

1. _____

What do you really need to know?

Determine what you want to learn about your application or system. Each type of test is run differently, and looks at your application in a different light. So, you'll need to run different types of tests based on what you hope to find out.

For example:

- | If you hope to discover how your application performs with little or no load in order to get a baseline, you will run a single user test.
- | If you hope to determine how your system will perform under normal expected load, you will run a load test.
- | If you hope to determine the breaking point, the point where your application either stops responding or responds so slowly that it is unusable, you need to run a stress test.
- | If you want to know if your application has memory leaks, you will want to run endurance or soak test.

2. _____

Decide on a number of users

If you are going to load test, how many virtual users do you want to simulate? In order to answer this, you will want to approximate how many concurrent users may visit your site, depending on the time of day.

Don't guess. Instead, leverage some of the data you already have. Talk to your marketing team and take



real traffic patterns from tools like Google Analytics, or engage your Operations team and use data from their Real User Monitoring (RUM) tools, which can provide insight into realistic scenarios to test. If you want to know some concrete statistics from a historical perspective, go directly to your analytics reports. You may even want to ask your engineers how many concurrent users they designed the application for, and your product owner for projected numbers, based on promotional activities. Plan to test that number and some percentage above it.

There are numerous ways you can find the number of virtual users needed to run a load test. We recommend the following formula to find the number of virtual users needed to run a load test.

$$\text{Concurrent users} = \frac{(\text{peak hourly visits} * \text{visit duration in seconds})}{3600}$$

You can find peak hourly visits and visit duration from an analytics tool, such as Google Analytics. Note that this formula should be used to estimate the number of concurrent users required to achieve a specific page-view rate at fully-ramped load. It should not be used to estimate the number of pages that will load within a specific time interval.

3. Study your analytics

Don't pretend to know how your customers use your application. The only way to truly understand your users is to study history (i.e. analytics). By studying your analytics, you will be able to create tests that are representative of your actual users, as opposed to tests that you think are representative of your users. In this regard, analytics are a tester's best friend.

4. Performance is a team goal

gather your team You need to involve a number of people in the testing effort, including: developer, network engineer, DBA, and business owner. All of these individuals have a vested interest in making the application successful, and each will approach the problem from a different angle.

The correct solution will not fall directly into one of these buckets, but will be a combination of two or more. Make sure each is aligned with the performance expectations and is available during testing to:

- | Monitor their area of expertise
- | Provide balanced feedback
- | Gain a sense of ownership for the health and performance of the application



5. --- Prepare your browsers

Use testing software that brings you as close to your actual users' experience as possible. You should be able to record your scenario in the browsers of your choice, but you also need to anticipate the browsers your users will most likely use. Consider the countries and regions where you anticipate high usage, and research the most used browsers.

You'll need to have these installed on your machine to begin testing. Then you need to make sure your load testing software emulates as closely as possible actual user behavior.

This includes:

- | Parallel thread processing
- | Think time
- | Multiple concurrent scenarios
- | Complex scenarios
- | Parameterization
- | Generating load from multiple agents (network/cloud)

In LoadNinja, you can leverage a real browser to run a test, so you know you are getting the most accurate test results as possible.

6. --- Be prepared to test your production application

While it is valuable to test your application when it is in a staging environment, this can leave some holes in your testing. There are a few reasons for this, including:

- | Staging environments are not often exact duplicates of production.
- | Staging environments are often accessible from only inside the firewall.
- | There is something to be said for testing the same system that you are gathering information about.

7. --- Set aside time to analyze results

You should be prepared to spend some time analyzing test results as a group (remember all of

those people that were present during testing?). Results need to be looked at to ensure bottlenecks/errors/weaknesses are understood and remediation is effective. Reach out to everyone involved and schedule adequate time.

8. --- Set aside time to make changes

Different remediation will have differing costs in terms of time. Remediation such as implementing a caching strategy, refactoring code, database optimization and hardware upgrades have a wide range of costs to implement in terms of both time and money. As an example, adding more hardware will require time to order the hardware, receive the shipment, test the new hardware, install software and data, test, install into the network, and test some more. This can be weeks or months of work. This is less of a problem if you are in the cloud. In which case, it takes less than a day. Many companies are opting to move to cloud infrastructures, offering an unlimited number of environments without the need for additional hardware costs and time constraints.



However, it's always advisable to load test applications that are accessible within your firewall as well. Some load testing tools, like LoadNinja, allow you to do both.

9. --- Plan an Agile testing methodology

Once you remediate, it is time to test again. The saying, "testing is a process, not a destination" is very true. Each time a bottleneck is uncovered and corrected, another one rises to take its place. It is important to plan an Agile testing methodology, whereby performance testing is baked into each step of the development cycle. Additional testing should be performed:

- | When code is modified or updated
- | When environment/infrastructure changes are introduced
- | When changes are made to the application server or DB server
- | When traffic spikes are anticipated

Now that you've taken the time to really prepare, load testing your application will help you continually improve your product and your business.

How role playing games can save your app:

It's easy to get swept up in role-playing video games. Who can resist the temptation to be anyone you want in a fictional world filled with unending excitement? Believe it or not, role playing games also offer valuable lessons for testing your application. In the next section, we look at how you can create scenarios in order to perform load tests, and why this needs to be a major aspect of both deploying and improving web applications.

10. --- How to record scenarios

The first thing to do is to determine the roles you will define for use in your test. A role is equivalent to a certain type of user that will visit the tested website, and the steps they will take while visiting.





If the tested site is a retail site, for example, you might have the following roles:

- | Browse and leave
- | Browse, add something to the cart, and check out
- | If your tested site is a restaurant site, your roles might look something like this:
 - Browse menu and find directions
 - Look at hours of operation and make a reservation

It's best to choose at least three of the most common pathways through your site, and add a few uncommon routes as well. Next, you need to break these roles down by percentage of traffic. A typical

retail site may have 95% of users browsing and leaving, and 5% (or less!) actually making a purchase (*Fig. 1*).

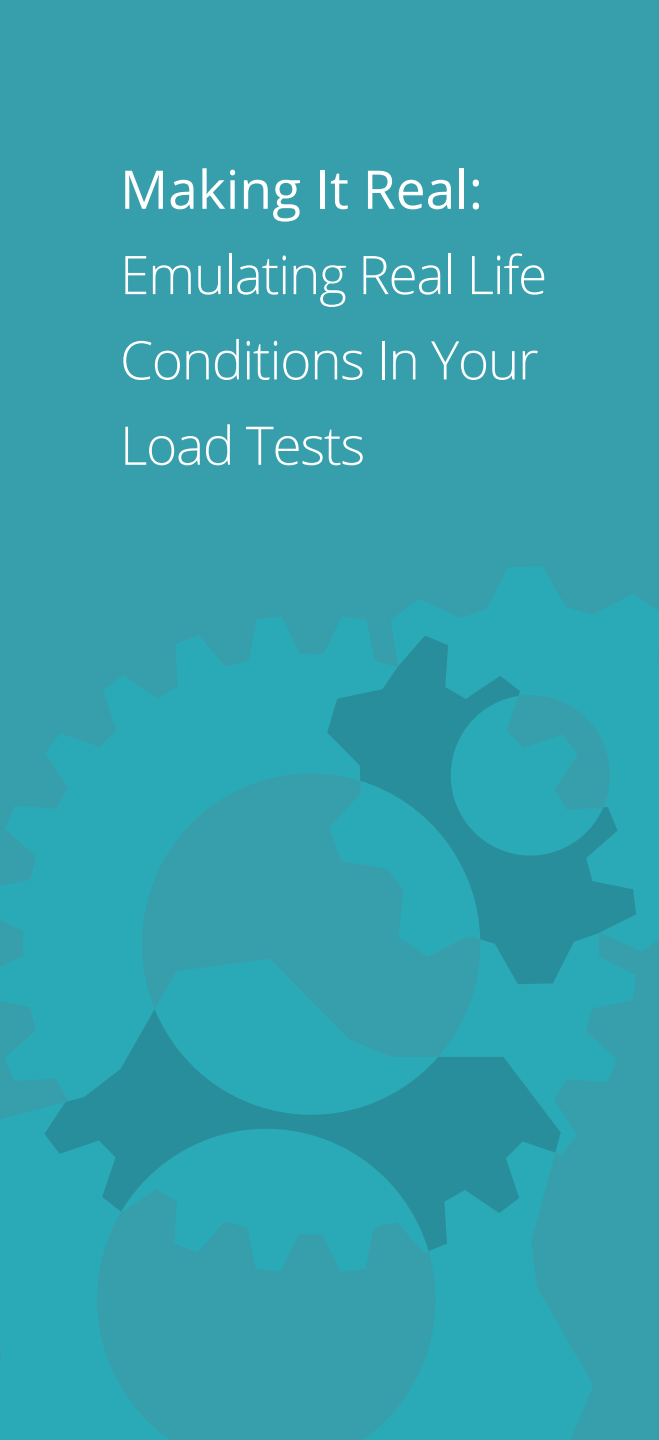
The combination of these two roles, or scenarios, will represent actual site traffic.

After each scenario is recorded, you need to verify it individually. This involves running a single virtual user for a single pass through the scenario. This step should never be forgotten.

Now you're ready to start testing. By recording scenarios that imitate actual user traffic, you're setting the stage to greatly improve customer experience and, if you're into e-commerce, get that percentage of purchases above 5%.

Fig. 1

Percentage	Role
95%	Browse and Leave
5%	Browse, add something to the cart and checkout



Making It Real: Emulating Real Life Conditions In Your Load Tests

Whether it's an elementary school math quiz, a college history exam, or a software development team's load test, we always want our tests to emulate real-life conditions. Otherwise, what's the point of testing?

In this chapter, we'll discuss ways you can ensure your load tests match reality. Some of the best perks about our load testing tool, LoadNinja, are the settings that aid the process of generating a realistic load test. Of course, like all load testing tools, you can specify the number of virtual users to be simulated, but you can also set certain conditions that easily create more powerful and reliable load tests. Plus, one of the biggest advantages is running your load tests in real browsers and getting test results with browser-based data. No matter which load testing tool you choose, make sure it allows you to set some version of these basic conditions.

Ramping up virtual users

When running performance tests, it's not desirable (and realistic) to start all virtual users at the same time. Starting all virtual users at the same time can create artificial bottlenecks in certain parts of the application — such as the login process. In Load

Ninja, you can configure the ramp up time and delay between user sessions, so you can increase the user number over time. For example: you start with a single virtual user and add one virtual user every two seconds until you reach a certain number of simultaneous users, and then hold that number for the duration of the test.

Setting load duration

In order for you to run a test with large numbers of virtual users, you will need to set duration for your test. By setting duration, each virtual user will execute its scenario and when it reaches the end it will start over – thereby maintaining the level of load. For example, if your scenario takes two minutes to be executed but you run a test for 10 minutes, the scenario might be executed five times by each virtual user. This function is part of the test set-up in LoadNinja, where you can select either a duration based test or a iterations based test. LoadNinja will simulate each virtual user for the number of times set by total iterations or for the duration that is selected.

Parameterizing tests

While recording a scenario, you may need to specify some parameters that will be used for further test



runs. For example, you can enter some search terms, user names and so on in the application's fields. However, it is not a good practice to play back a test with the same recorded data for each user as it does not simulate the real-life conditions. To solve the problem, LoadNinja allows you to parameterize your load test using real data. Data-driven tests empower your team to create tests that reflect realistic scenarios, and by leveraging this feature, the requests can use the provided data during test runs.

Replicating Browsers, network connection bandwidth and browsing speed

Real users visit a website using different browsers, the bandwidth of user's Internet connections can vary significantly, and they spend different amounts of time on each page. It's important to keep all of these in mind when you're configuring your tests, so you know that the tests you run will be helpful in gauging how your application will perform in production.

In LoadNinja, you record, replay, and run every test in a real browser. This way you know you're accu

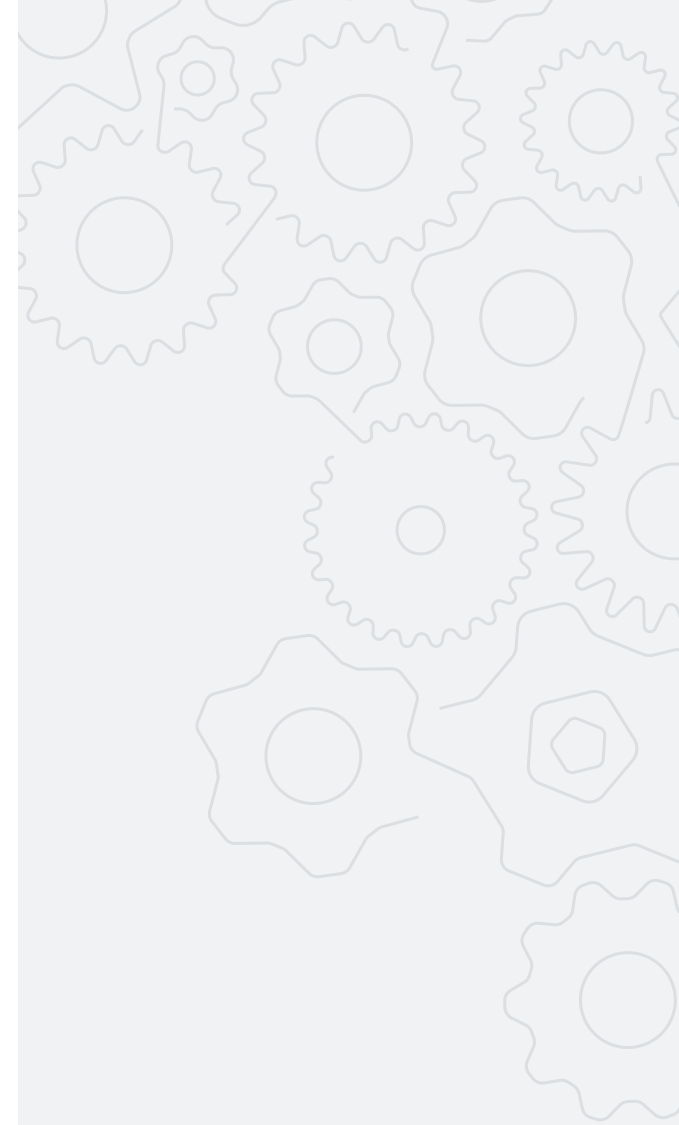
rately measuring your application performance. With this advantage, your team can spend more time testing & analyzing and less time reconfiguring the original test scripts to ensure they playback properly.

You can also specify think time for each tested page (we call this think time, as it simulates the time when a user is viewing the page and thinking). You can easily randomize the think time for simulated scenarios to better emulate real user activity.

Analyzing data in Load testing: What you need to know

More often than not, developers and QA managers come away from a trial of load testing software with little more than the number of users that will crash their system. Unless they have a professional load tester on staff, most development teams don't have the resources, or knowledge to garner all they could from their load tests.

That data is wasted because the person running the tests is unable to apply it to an application's performance. Luckily, improvements in graphics and UIs have made interpreting data much easier — if you know what to look for.



Here's a list of the most important results in load testing and how you should be working with them.

Page load time

You need to know the average page load time for each page in your scenario. You might have a strict Service Level Agreement (SLA) that mandates how quickly pages must load, or may just want to know what this number is. It is also important to know if one page takes longer than others to load— this indicates a bottleneck in your application.

Response load time

Just knowing page load time is not enough. If a page is slow, you need to know why. Being able to look at average response times for each response gives you a detailed look into where the time is being spent.

Errors and warnings

You need to know which errors and warnings were generated and at what level of load. This is especially important information to see in chart format. It

is important to see which errors and warnings are generated and be able to see how that changes as load increases. A common error at high levels of load is “Server Error 500s.”

Navigation Timings

Understanding what your end user experience and why is key in identifying performance issues. Some of these metrics include redirect time, connect time, first byte time, response time, DOM load time, and event time. It's important to understand what your performance benchmarks (or competitive benchmarks are) for these metrics to make sure that the experience you're delivering through the browser meets your standards.

Request and response throughput

It's important to see the amount of data going to and coming from the tested system. This is espe-

cially important in a case where load is increasing, but bandwidth reaches and maintains a plateau. In this case, it becomes apparent that bandwidth is being throttled at some point in the process, possibly at the firewall.

Hosts

Because so many of today's websites call out to a plethora of additional hosts for things like content delivery networks, ad servers, analytics servers, social media and syndicated content, it is important for these sites to be enumerated in your reports. It's equally important to be able to view all of the calls to a particular host. If a host is called from your pages, the response time for those requests will add to the time it takes your pages to render.



The Dos & Don'ts of Load Testing

As you get ready to implement load testing to your performance strategy, there are a few dos and don'ts you'll want to keep in mind.

Dos:

- ✓ **Record tests** with an end-user in mind. Map out user journeys that are realistic, comprehensive, and critical to business functions. Understanding that your application can handle what a user would realistically do is key in ensuring it will stand up in production.
- ✓ **Generate load** from different servers than those that host your application. Serve them from different environments so you can get the best view of performance without accidentally altering results.
- ✓ **Start recording** a new scenario from the web browser's start. If you start recording a scenario after you connect to the tested web server and open a few web pages, the playback of the scenario will fail. This will happen because the recorded traffic will not reproduce the authentication procedure, and the tested web server will ignore the simulated requests. (Note: In LoadNinja, this is

not required, as we test in real browsers. However if you're relying on an emulator this may be important).

- ✓ **Parameterize scenarios** to simulate more realistic load on the server. Parameterizing scenarios involves replacing recorded parameters in the requests with variable values. This way you can make virtual users send user-specific data to the server. These data-driven tests will give you a more realistic view of how your application will perform when a unique group of users interact with it.
- ✓ **Verify user scenarios**. Before creating tests on the basis of a recorded scenario, make sure that the scenario is executed successfully for one virtual user. This can help you identify bottlenecks of the scenario and eliminate problems which are not related to the number of virtual users and additional testing conditions.
- ✓ **Arrange user scenarios** in your tests so that critical functionality is tested first.





Don'ts:

- ✗ **Do not run tests in real environments.** A real environment can have other network traffic, and this may affect test results. To avoid excess data transfer, use a test environment that behaves in the same way as the real environment except that there is no other traffic usage. (Refer details on 'prepare' stage to perform load test in production).
- ✗ **Do not try to crash the tested server.** The goal of web server performance testing is not to break the server, but to identify web application performance bottlenecks under various loads.
- ✗ **Do not overly stress the client test systems.**
- ✗ **Do not use zero think time.** Make sure that think time in your test is based on real-life conditions. Using zero think time does not provide realistic user simulation and puts an abnormal load on the tested server. However, omitting think time can help you determine bottleneck issues.



About LoadNinja

LoadNinja by SmartBear allows you to quickly create scriptless sophisticated load tests, reduce testing time by 50%, replace load emulators with real browsers, and get actionable, browser-based metrics, all at ninja speed. You can easily capture client-side interactions, debug in real time, and identify performance problems immediately. LoadNinja empowers teams to increase their test coverage without sacrificing quality by removing the tedious efforts of dynamic correlation, script translation, and script scrubbing. With LoadNinja, engineers, testers and product teams can focus more on building apps that scale and less on building load testing scripts.



SMARTBEAR
LoadNinja

[Start Your Free Trial](#)



SMARTBEAR