

Creating Load Tests in LoadNinja



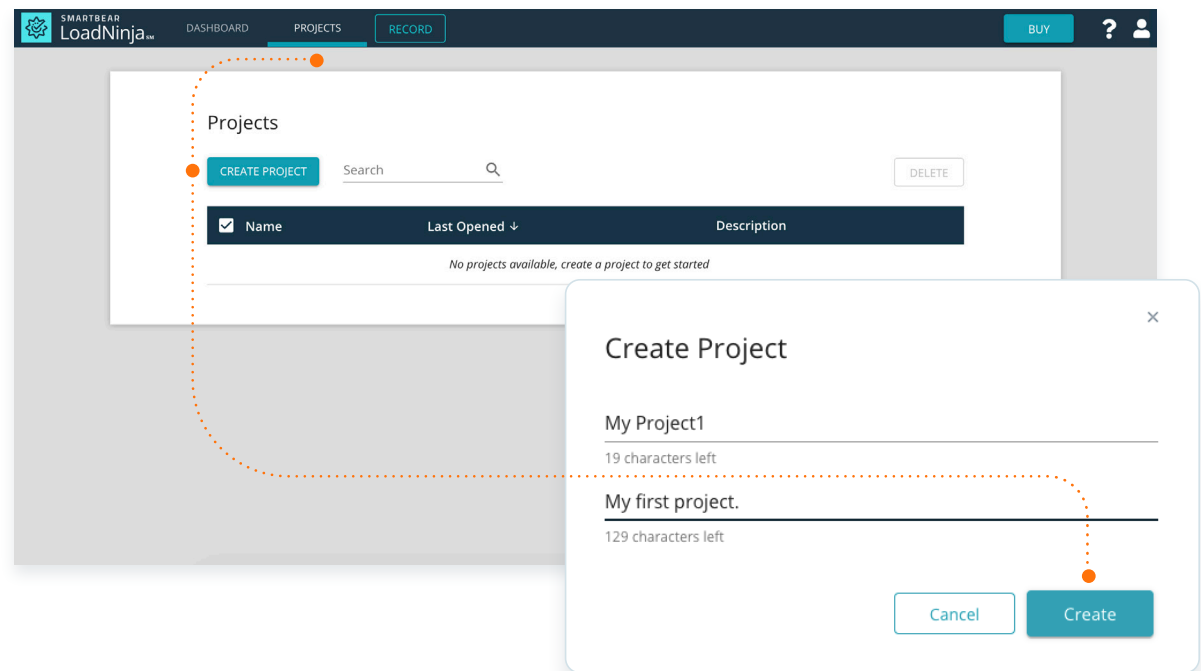
If you don't have an account already, you can sign up for a free trial to follow along or try it on your own application.

[Start Free Trial](#)

LoadNinja makes it easy to build load tests in a browser-based interface with no or minimal code. Before getting into test automation, let's get started by recording a simple test, configuring the load test, and seeing how to read the results. We can then implement this test scenario into Jenkins to automate the process.

Start by creating a new project:

1. Go to Projects and click Create Project.
2. In the dialog box, specify the project name and description.
3. Click Create.



The next step is creating a test script:

1. In the Project, click Record.
2. In the dialog, specify whether the application is Internal or External.
3. Specify the URL of the web page where the test will start and the desired screen resolution. Optionally, configure any advanced options.
4. Click Start Recording.
5. Interact with the web application in the same way that an end user would by clicking the links, entering text, or scrolling. You should be sure to use the same cadence as a typical user to mimic a realistic workflow. These steps will appear in the left sidebar as you go, where you can edit or remove them, as needed.
6. Click Stop.
7. Edit any of the steps in the left sidebar to fine-tune the test.

Note: LoadNinja simulates script events using hard-coded parameter values during the editing stage by default. However, you can use a databank (e.g. CSV or

TXT file) to expand test coverage with more parameters. Simply upload the file and assign each column to the associated input. This lets you test multiple search queries, user logins, or other details to create broader test coverage than static hard-coded parameters.

LoadNinja also makes it easy to test client-side interactions and automatically handles encryption, authentication, authorization parameters, and many other complex scenarios. Since every test is recorded and replayed in a real browser, our platform automatically supports third-party frameworks like React, Angular, and HTML5. These are issues that can require significant extra scripting using protocol-based load testing platforms, such as JMeter or Gatling.

Next, create a scenario and run the load test to ensure that it runs properly:

1. Click on Save and Create a Load Test Scenario.
2. Define the number of concurrent users and test duration.
3. Optionally, click on More Config Options to configure different settings, such as the ramp-up time or delay between iterations.

Finally, you can see the results and debug any issues:

1. The Scenario will show the status (e.g. running or finished) and the initial results.
2. The Charts tab provides accumulated results in a graphic form.
3. The Statistics tab provides the results of individual scripts in the scenario.
4. The VU Inspector tab lets you view desktops of remote cloud machines where virtual users are working.
5. The VU Debugger tab lets you see information on errors that occurred during the test run. The buttons on the right let developers dig deeper into each error.

The entire process is much more streamlined than using JMeter, Gatling, or other load testing tools that require a lot more scripting and configuring to properly measure modern applications. In addition, the results are much more accurate and actionable since they use real web browsers rather than measuring protocol level traffic.



Executing on Load Test Automation

LoadNinja makes it easy to accomplish load tests in a fraction of the time that it would take with a conventional load testing platform. Fortunately, it's just as easy to implement these load tests into a continuous integration environment, like Jenkins, that builds and tests software projects continuously after each commit or merge. That way, any new code changes that break the software are instantly identified and flagged before reaching production.

Start by downloading the LoadNinja plugin for Jenkins:

1. In Jenkins, go to Manage Jenkins and then Manage Plugins.
2. On the Available tab, type "LoadNinja" into the Filter field and check the box next to LoadNinja Load Testing Plugin when it appears.
3. Click Download now and install after restart or Install without restart.

Next, create a new Jenkins project:

1. Click on New Item in Jenkins to create a new project.

2. Enter the item name and select Freestyle Project and click OK.
3. Add a Build step on the following page and select the LoadNinja Plugin.
4. Input the LoadNinja apiKey and scenarioId. Optionally, set any pass criteria for test errors or step durations. The API key can be found in the LoadNinja User Settings, while the scenario ID can be found at the end of the scenario's URL.

Note: You can also add LoadNinja as a build step in an existing project. For example, you may want to add performance testing to your functional test suite that runs before a merge to a production branch on git.

Finally, configure the test results:

1. Click on Add post-build action in the Post-build Actions section.
2. Select Publish JUnit test results report to receive an XML report. You can specify the name of the target file for the reports in the appropriate fields.
3. Click Save.



Performance tests can be run at any stage of the development process. Since it's resource intensive to run after each commit on a development branch, most development teams run the tests before each merge with a production branch to ensure there are no bottlenecks. Jenkins makes it easy to automatically trigger tests through integration with git or using tokens, putting developers in control of when and how they run the test suite.

If you're not using Jenkins, LoadNinja provides a REST API for integrations with other continuous integration platforms, such as CodeShip or CircleCI.

The screenshot displays the Jenkins web interface. At the top, there's a black header with the Jenkins logo and name. Below it, a green navigation bar contains links: 'New Item', 'People', 'Build History', 'Manage Jenkins' (highlighted with a red dot and a dashed line), 'Credentials', and 'New View'. The main content area is titled 'Manage Jenkins' and lists several configuration options, each with an icon and a brief description:

- Configure System**: Configure global settings and paths.
- Configure Global Security**: Secure Jenkins; define who is allowed to access/use the system.
- Global Tool Configuration**: Configure tools, their locations and automatic installers.
- Reload Configuration from Disk**: Discard all the loaded data in memory and reload everything from file.
- Manage Plugins**: Add, remove, disable or enable plugins that can extend the functionality. A red triangle icon indicates 'There are updates available'.
- System Information**: Displays various environmental information to assist trouble-shooting.
- System Log**: System log captures output from `java.util.logging` output related.

On the left side of the 'Manage Jenkins' section, there are two expandable panels:

- Build Queue**: Shows 'No builds in the queue.'
- Build Executor Status**: Shows a list of executors:
 - 1 Idle
 - 2 Idle

How to Read Load Test Reports

Most people limit the load testing to average response times, but there are many other metrics that are important to consider. For example, high error rates can degrade the user experience even if average response times are low. The number of concurrent users is also an important consideration since response times depend on the load. And finally, none of these metrics matter unless developers can identify and fix the bottleneck responsible.

LoadNinja focuses on four key elements:

1

Duration is the average duration for running each script and step. These numbers include both the time it takes to perform the actions, as well as the think time intervals that account for user behavior.

2

Navigation timings are the average time it takes to move between the website's pages and locate and identify objects there. For example, you can see redirect time, DNS time, connect time, first byte time, response time, and DOM load time.

3

Error count is the number of errors for each script and step. These errors may include connection timeouts, missing elements, or validation failures for assertions that are created within LoadNinja.

4

Success rate is the number of successful iterations. In other words, the success rate is the percentage of steps within a script that pass successfully during the test run.

You can also access the performance metrics for LoadNinja servers, which helps identify cases where the test infrastructure is the cause of errors that appear in the test results. You can also decide how many virtual users can be engaged in load tests since LoadNinja's performance differs depending on the complexity of the load testing scripts.

Navigation timings are especially helpful because they can help pinpoint the true underlying bottleneck. For instance, you may notice that the DNS time for a particular step is taking a long time, which could suggest a problem with your DNS





provider rather than an underlying application issue. LoadNinja's inclusion of DOM load times also sets it apart from other load testing platforms that only measures server requests and responses.

When it comes to continuous integration, it's a good idea to set minimum thresholds for error rates or durations in order for the test suite to pass. You should also ensure that you're testing with an appropriate number of virtual users for your expected loads. The goal is to identify any performance bottlenecks at expected peak loads, not crash the application under an unrealistic load that you don't expect to see in production.

A key benefit of LoadNinja is the ability for developers to debug bottlenecks in real virtual browsers as they occur. The VU Debugger provides full access to the test page's DOM via the document object in a JavaScript terminal. Developers can easily access web page elements in the same way as regular JavaScript on web pages. They can also view network traces to identify the source of issues that virtual users encounter.



Managing Tests with Zephyr for Jira

Test automation is critical for software development teams that want to move at a high velocity and increase feedback while maintaining or optimizing quality. Over time, test suites can become unwieldy and difficult to manage, particularly for product owners that aren't involved in the low-level creation and maintenance of tests. Test management solutions can help ensure collaboration and traceability throughout the development lifecycle.

Zephyr for Jira is the most popular suite of test management tools for Jira designed to optimize the speed and quality of software testing, empowering teams with flexibility, visibility, and insights. From test plans to quality metrics, the platform makes it easy to keep everyone on the same page with test automation and integrations with popular continuous

integration tools. You can even use advanced search queries with ZQL to find specific tests within a project.

It's easy to integrate LoadNinja's load tests with Zephyr for Jira and Jenkins to provide access to these collaboration tools and capabilities.

Start by downloading the Zephyr for Jira plugin for Jenkins:

1. In Jenkins, go to Manage Jenkins and then Manage Plugins.
2. On the Available tab, type "LoadNinja" into the Filter field and check the box next to Zephyr for Jira Plugin when it appears.
3. Click Download now and install after restart or Install without restart.



Next, add the post-build action to Jenkins:

1. In Jenkins, navigate to the Item/Project and click Add Post-build Action.
2. Select Publish Test Results to Zephyr for Jira.
3. Add the Jira URL and Project Name.
4. Click Save.

In Zephyr for Jira, you can click on Tests to see the results of the tests. In Cycle Summary, you can see all of the previous tests that ran in the past and whether they passed or failed. Developers and stakeholders can log in to Zephyr for Jira at their own leisure to view these test results without test engineers having to send over reports to hold meetings. Any problems can be easily converted into bug reports and flagged for follow-up.

JIRA Dashboards ▾ Projects ▾ Issues ▾ Tests ▾ [Create issue](#)

IronClad
Key: IC · Lead: Zephyr Admin

Summary
Issues
Road Map
Change Log
Test Summary
Test Cycles
Popular Issues
Versions
Components
Labels

Test Cycles

▼ **Cycle Summary**

Select Versions:

Functionality
Started On: 18/Mar/12

Major features of Iteration 1 (all 8 stories) will get tested in this cycle
Build: 766
Environment: **QA1**

ID	Status	Summary	Defect	Component
IC-1	FAIL	Verify the app can be installed from the AppStore	-	Main screen
IC-3	PASS	Verify the app can be launched	-	Main screen
IC-6	PASS	Verify app starts cleanly	-	Main screen
IC-7	BLOCKED	Verify app can be closed	-	Main screen
IC-8	FAIL	Verify all the buttons on the app show up	-	Main screen
IC-9	PASS	Verify all the images show cleanly	-	Main screen
IC-10	UNEXECUTED	Verify that the app can be upgraded	-	Main screen
IC-12	PASS	Verify app can be dimmed	-	Main screen
IC-13	PASS	Verify app can put in hibernation	-	Main screen
IC-14	PASS	Verify that the app can have different backgrounds	IC-11	

Regression
Started On: 23/Mar/12

The full set of regression tests and more
Build: 766
Environment: **QA2**

Final Sanity Tests
Started On: 18/Apr/12

Final tests on Staging before we push this out. Test on all major browsers
Build: 766
Environment: **Staging**

