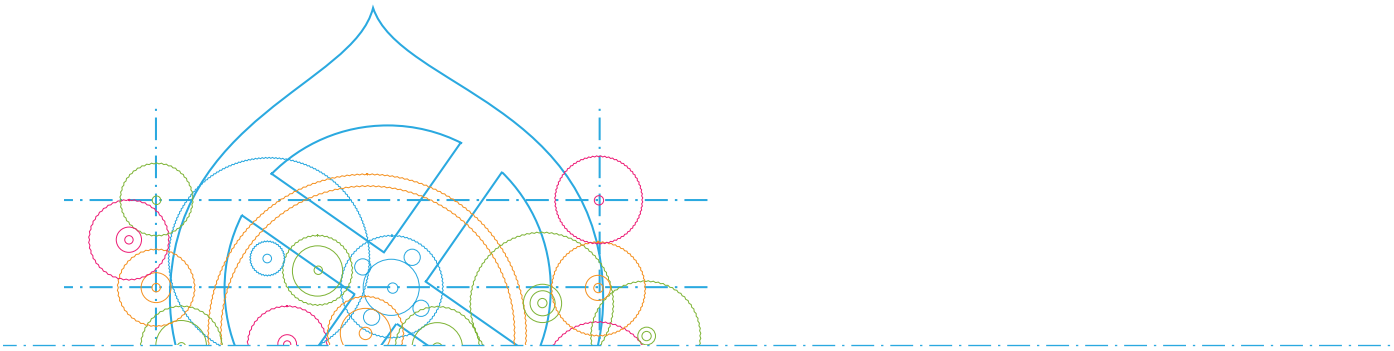
Acquia®

TECHNICAL GUIDE



Best Practices For Load Testing

Want to understand how your website handles traffic? Need to ensure your new site will stand up to the traffic spikes on your current site? Want to ensure your application will keep responding under the load of an upcoming marketing promotion?

Load testing your application is the best way to handle these scenarios.

There are several different load testing platforms out there, from full SaaS providers to do-it-yourself (DIY) options. And there are different ways to approach load testing philosophically: Should you test your site on a single server to understand more about how it scales? Or should you test your site on a full production replica environment? Let's dive into these questions.

Load testing is really about risk mitigation. How well you test your application is a direct response to how well you tolerate risk, how critical the site is, how dramatic the spikes in traffic, etc. It can be very difficult to completely simulate real-world conditions on your site. You can review your analytics software in depth, test all the top hit pages on your site, and still potentially miss that one custom admin page that ends up blocking the database queries from your webs. The closer you can get to real world scenarios the better, but be aware that no load test can catch absolutely everything.

There are two primary ways to load test your site: application performance profiling and full-stack simulation testing. But before you begin load testing, there are fundamentals that need to be addressed, such as determining what to test and what to monitor.

Determining What to Test

The first thing you will need to do is to sort out what you need to test. If you are load testing an existing site or updating an existing site, then looking through past analytics to identify traffic patterns is your best bet. If this is a brand new site, you may need to look at third party traffic sources such as Alexa to get an idea of what traffic may be like when the site launches. You also need to ask yourself some fundamental questions such as:

- Do different types of traffic affect my site differently (e.g. anonymous users versus authenticated)?
- Are users broken into logical groupings (e.g., viewers versus contributors)?
- What do users actually do on my site?
- Are there transactional flows to consider modeling (e.g. on an eCommerce site)?

Typically with Drupal, anonymous users are "lighter" on the server than authenticated users because of the various caching layers that should be in place. Drupal has the ability to cache entire pages for users and more complicated caching architectures involve reverse proxy caches (such as Varnish) and more specialized caching strategies (such as block, entity, and/or view caching). Also keep in mind that anonymous users still may bypass these caching layers if they are doing special things like submitting forms, adding products to their shopping cart, hitting the site with lots of unique query strings, etc. You will want to make sure that you identify these more complex scenarios and write them into your tests. We've seen sites that load tested admirably and then fell over once they were live because an inbound link was sending a random query string with every request and effectively bypassing caching. You can never be too careful in understanding how traffic is actually getting to your site.

Monitoring and Setup

You will need good monitoring on your server(s) as a prerequisite to running a load test. For Drupal, you will want a php profiler like xhprof and/or New Relic. You'll also benefit from using a query digest type tool like MySQL's slow query log or Percona's pt-query-digest to aggregate what queries are running on your database server. You also may find it useful to tail your load balancer's access logs as well as more standard web server logs like Apache's access log and php's error log. You'll want to monitor overall server metrics such as cpu utilization, memory utilization, disk I/O, etc. And finally you'll want some charts to measure these metrics over time. Using statsd with a visualization tool such as Graphite can be a great way to gain deeper insight into the performance of your servers. Note that this paper isn't intended to be a monitoring setup tutorial and detailed monitoring how-to's can be found easily elsewhere online.

In addition to monitoring, there are other "gotchas" that you should plan for. These include things like: ensuring that system email isn't being sent, working with test user accounts instead of real users, and making sure you account for third party integrations. If your tests trigger solr searches, simulate ecommerce checkouts via payment gateways, or include some sort of analytic or statistics gathering, you will need to have a plan for how these services will handle the load and the extraneous data.

If you run load tests often, it can be helpful to write small scripts to implement your ideal monitoring. A load test is only as good as its monitoring. If you don't have good monitoring in place, you won't have any way to analyze what's going on during the tests.

Now that we've figured out what to test and how to monitor the test, let's address two general approaches to load testing your application.

Application Performance Profiling

Application performance profiling is about taking a website, putting it on a known, well performing web server configuration (usually a single server), and throwing small amounts of load on it in order to see how it responds. It's generally cheaper and simpler to do this type of testing, and when done correctly it can be very effective in identifying key performance issues. Because this doesn't require much (or any) additional hardware, you can afford to run this type of testing during your development cycle. Having ongoing metrics of how your application's profile has changed over time can be very handy in identifying when poor performing code has been introduced (or conversely when performance improves as a result of new code).

You will need a control site on the same hardware that can be tested as well in order to create a good comparison. In our case, this is typically a stock Drupal site with some content and some views or potentially a demo Drupal distro site. You're likely not trying to stress test an entire stack or measure complex network latency. Rather, the goal is to get a good understanding of how your application performs compared to other simple and complex sites. If you are using a DIY tool (such as jmeter or siege), you can make sure the load generator is close to the application server from a network standpoint.

For this type of testing, it's generally a good idea to go for a smaller number of concurrent users. You will want simple common user journeys in your testing, so the test scripts won't be comprehensive models of your actual traffic patterns. You're interested in seeing how the application handles a predictable set of users (say 50 or 100) compared to it's peers.

THE FOLLOWING ARE THE PROS AND CONS TO THIS APPROACH:

Pros:

- Cheaper and easier to do than full-stack simulation testing
- Doesn't require much hardware
- Can be done pretty quickly once you've setup the scenarios
- It's easy to zero in on specific application issues since you aren't tinkering with things like MySQL settings or LAMP stack tuning
- Can be used on in your development workflow with a CI tool such as Jenkins to monitor how your application handles load as new features and new code are deployed

Cons:

- Isn't an exhaustive test and can't guarantee that your site can hit a target number of concurrent users
- Isn't intended to test your full infrastructure (reverse-proxies, CDN's, etc.)
- Doesn't work well for testing really high-traffic sites

Full Stack Simulation Testing

In contrast to Application Performance Profiling, running load tests against a full replica of your production stack is a more comprehensive way to load test your application. Full Stack Simulation Testing is simpler to do if you are hosted in a cloud-based datacenter, since you can spin up a replica of your production configuration for a short time to run the tests and then spin it back down when you are finished. It is still possible to do Full Stack Simulation Testing in a non-virtual datacenter, but you will need to consider the impact of adding new hardware and testing through shared components like load balancers or network switches. It is also important to have every piece of the stack in place, so if you use a CDN, WAF or reverse proxy you will want to run your tests through those.

This approach also requires a more rigorous set of tests that mimic actual user journeys on your site. You will want to take a close look at your existing site analytics, break users into groups or personas, and create representative test scripts that simulate a real user's journey on the site. It is helpful to also variabilize and randomize your scripts so that users are not always hitting the same pages on your site. If administrators frequently login to the production web servers to perform editorial or administrative tasks, it is also helpful to model these users in your scripts. You will also want to run any scheduled processes (such as Drupal's cron) during the tests so that you can see how backend process may affect the servers under heavy load.

For this type of testing, you are looking for peak traffic handling, so your tests will start with a small number of users and ramp to a peak. If you have analytics for a previous version of the site, you will want to look for peak hours of traffic. Typically, it is recommended to shoot for at least 25 percent more load than your highest previous one hour spike in your tests.

THE FOLLOWING ARE THE PROS AND CONS TO THIS APPROACH:

Pros:

- Effectively simulates "real world" traffic scenarios
- Very useful when you want to see if your production stack can handle a certain volume of users
- Reduces launch risk considerably
- Exposes performance bottlenecks that only surface under extreme conditions in both the application layer and in the underlying hosting stack

Cons:

- Can be much more expensive that application performance profiling
- It takes more expertise and time to create and run this type of load testing

Putting it all together

So what do these approaches look like in practice? If you've got a low profile project, you could get away with doing Application Performance Profiling toward the end of your build to make sure there aren't glaring problems with your site's performance. If you have a high profile project or a high traffic site, it's a great idea to connect the Profiling concept into your CI solution (such as Jenkins) so that you are constantly evaluating performance as you build. Make sure you save time toward the end of your project to do Full Stack Simulation Testing to ensure that your critical site will stand up to the expected traffic. This combination of ongoing performance testing during development and stress testing can help ensure a hassle-free launch.