WP engine®

Preparing for peak traffic: A load test guide.

By Tony Le, WP Engine Sr. Solutions Engineer

WHITE PAPER

Table of Contents.

Introduction.

With the holidays approaching, most marketing teams are prepping for impactful, revenue-driving campaigns. Some might even be preparing for the launch of a new product on Black Friday. If that's the case, a lot of behind the scenes work and build-up has gone into the launch.

When it comes to ensuring success on launch day, performance should be paramount. All the thought, strategy, and preparation behind the campaign will be for nothing if the site can't handle traffic during launch. With revenue on the line, it's best not to play the guessing game; knowing the amount of visitors your site can handle should be a pivotal part of your pre-launch process.

The question of concurrent visitors isn't an uncommon one. In fact, it's top-of-mind for many WP Engine customers hoping to win online during the busiest online shopping events of the year. Unfortunately, the answer isn't always straightforward. The amount of load a site can take depends, for the most part, on two things: infrastructure and application code.

Load testing, which is performance testing that simulates real-world load on any software, application, or website, can help answer the question: "how many people can visit my site at once?" Proper load testing can help site owners assess things like scaling capabilities, lifecycle hooks, security risks, automatic code deployment, health checks, and target tracking.

In this informative white paper, we'll break down the basics of load testing, why you should load test, best practices, and how to get started load testing your site before the holidays.

What is a load test?

A load test is the process of putting demand on a system with the purpose of determining how it will function. Most commonly, a load test is performed to simulate peak traffic with a high number of concurrent users. In other words, how does the site perform under heavy load?

We often get asked, "how many concurrent users can this environment handle?" And the most common answer we give is that it depends. On a lot of things. If we boil it down, there are three major variables that could go wrong during a major event:

Firstly, it could be the infrastructure and platform solution you're using. It could be that your infrastructure is simply underpowered or under-architected. There are also many services and applications that go into powering a WordPress site (at its most basic these could be Linux, Apache, MySQL and PHP).

These could be poorly configured and therefore result in poor performance and downtime.

Secondly, the problem might not be infrastructural at all. It could be that the WordPress application is particularly inefficient, poorly written, or highly dynamic. If the site isn't tuned, all the infrastructure in the world won't help. Vice versa, if the infrastructure/platform is slow, that negates the speed of a finely tuned WordPress site. It's important to address both as you load test.

Finally, the third critical variable is to test the right things. There's no point testing functionality on your site that isn't used by a reasonable segment of your visitors, but we'll get to that in more detail later.

Why load test?

From a business and technical standpoint, it's important to load test for many reasons, with the underlying theme of mitigating risk.

Functional tests don't reflect the real world

Before going live, your plugins, themes and WordPress core may pass a functional test but under normal traffic or heavy load, the production environment could see degraded performance. Load testing can identify when and where the breaking point occurs. With more visibility, you can proactively fix issues before they become a bigger problem and impact you when it matters most.

Downtime is expensive Every minute counts

According to Blazemeter, one minute of downtime during Black Friday costs an organization $4,700 on average. Gartner cites network downtime costs, on average, of $5,600 per minute.

Within the opening minutes of Prime Day in 2018, shoppers reported error messages and broken landing pages on Amazon.com. In the U.S., Amazon is estimated to have lost $72.4 million in revenue based on 63 minutes of downtime. That's about $1.15 million of lost revenue per minute.

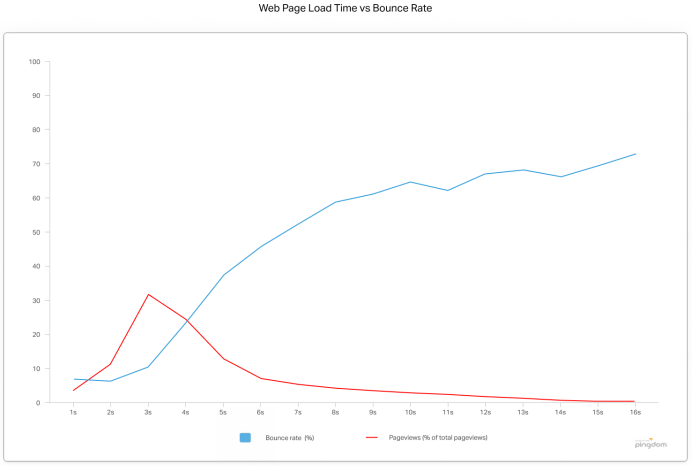
In addition to lost revenue, there are indirect costs associated with downtime as well. These include:

- **Lost productivity:** Employees must set aside their core role and responsibilities to focus on disaster recovery. Their focus shifts from revenue-impacting work to reactive troubleshooting and putting out fires.
- **Recovery costs:** There are costs associated with recovering your production environment such as potential loss of data. Additionally, in times of urgency, if you can't solve the problem with the current resources, you may need to invest in third party integrations or additional infrastructure to alleviate the current pain temporarily.
- **Intangible costs:** The cost of damaged brand reputation is less concrete. However, consider the long-term consequences of downtime and how customers may perceive your brand in a highly competitive market.

Needless to say, planning effectively for big events shouldn't be understated or left until the last minute. Load testing is often overlooked in the process and leaves your company in a tough position when issues arise.

Users don't have time

According to Pingdom, 50% of users will leave if a page takes longer than 6.5 seconds to load. There's a direct correlation between increased page load time and bounce rate percentage.



(The correlation between pageviews and bounce rate%.)

Load testing will help uncover hidden issues so that your team can provide an optimal user experience for visitors. This also prevents visitors from getting frustrated and going to a competitor's site for the same content or product.

Best practices.

Before we dive into the how-to, let's address a few best practices:

Test early and test often

- The earlier you can test, the more time there is to fix any bugs or address any bottlenecks. There may be some easy optimizations that can be done which will save time in case larger issues need focus down the road.
- For normal day-to-day activity, we recommend performing a small load test after any major site changes—especially those that focus on functionality. That way, you can see how the changes might affect your daily users.
- For high profile events, a load test with larger volumes of concurrent users may present different results than a normal traffic scenario. As these events typically involve many stakeholders, performing these tests well in advance can inform business decisions. Meanwhile, the technical team can continue optimizing if results are subpar.

Example: Does it make sense to have a fancy widget that dynamically generates real-time data upon every page load? This could mean investing in more backend resources to support that user experience. Perhaps, it's optimal to temporarily disable this functionality before the event.

Don't test on production

- You wouldn't want to affect your everyday users and daily revenue. If you're going to put your site under heavy load and simulate maximum traffic, it's best to do this on a separate environment that is as similar to the production environment as possible. This includes theme, plugins, data, third party integrations like CDN, and DNS. On that note, you may want to create a unique testing domain versus using our temporary wpengine url to replicate DNS performance.
- If you can't test on a separate environment, then we highly recommend testing during a window when traffic is minimal. For most businesses in the U.S., that may be around 3 am to 4 am. It's best to determine this window using an analytics tool like Google Analytics.

What kind of test?

- There are many different types of tests including:
    - **Stress testing** - also known as fatigue testing, this test reveals how the system behaves under highly intense loads. How does the system recover?
    - **Spike testing** - a type of stress test evaluating how the system performs during a sudden increase in workload. Fast ramp-up. How does the system respond to spikes?
    - **Soak testing** - tests the system over a long period of time through a slow-ramp up. Evaluates sustainability. How does the system respond throughout the day?
- For this guide, we'll focus on stress testing to understand the limits of what your environment can handle. By finding the breaking point, you will be able to better understand capacity and strategize scaling efforts to handle periods of high demand, giving you the confidence and peace-of-mind that things will run smoothly when it's time to hit the big stage.

Define KPIs

- To gain the most value out of the load test, you'll want to define metrics and parameters for success—and failure.
- What is an acceptable response time, error rate, number of transactions passed/failed, requests per second?
- Testing against your KPI criteria will also help evaluate if they are realistic or need adjustment.

Monitoring tools

- **Google Analytics** - as the test is in progress, run the current metrics against Google Analytics. Are the numbers of active users consistent with virtual users tested? Are there any drops in page views per second?
- **Infrastructure/platform monitoring** - provide a notice in advance that you are testing the environment to ensure TechOps alerts aren't triggered without need. If you're a WP Engine customer, you'll want to prevent our security layer from blocking requests, which means you'll need to whitelist

the IPs of the traffic source. Depending on the provider, the service may be able to provide digestible analytics that you might not otherwise have visibility into.

- **Application Performance Monitoring** - using a service like New Relic APM provides visibility into the WordPress site under heavy load.You may set up alerts when certain thresholds are reached and keep track of KPIs including requests per minute, response times, and apdex score.
- **External Uptime Monitoring** - services like Pingdom and Uptime Robot offer external uptime monitoring services which regularly ping your website to ensure it is reachable. As a last line of alerting, use these tools as you normally would for day-to-day monitoring.

Third party services - load testing

There are many options in the market for load testing. Since planning and executing the test takes time, choosing the right platform is important to maximize ROI. Several things to consider when choosing the best service that works for you include usability, supportability, realistic user scenarios, repeatability, and cost.

- Blazemeter
- LoadNinja
- Loader.io
- Artillery.io

Where's the tipping point?

The big question is...when and where does the breaking point occur?

"Industry standards suggest that systems are considered under load if 80% of resources are being utilized, and you should test at least 20% over your expected peak." -Blazemeter

After a load test, you can view the results to determine where the environment is saturated with requests. The test environment may reach capacity at the point in which hits/sec stops increasing and plateaus as virtual users increase. This indicates the system can no longer support a larger number of concurrent requests and the environment is capped at X virtual users.

Image by Blazemeter

At this peak load time, you can use tools like Application Performance Monitoring and load test results to cross reference data and see how performance may be affected. If the results are subpar, WP Engine customers can reach out to our Support team for help understanding which parts of the application experienced the most stress.

Creating a realistic scenario

- It's best to start with engaging your marketing, product, and technical teams to understand what the user journey will look like.
- If this is an event that has happened in the past, you can pull historical metrics to understand how users behaved on your WordPress site. Tip: You can use Google Analytics to understand the most popular pages visited, time spent, and bounce rates. WP Engine's Content Performance can provide visibility into this as well.
    - If this is an ecommerce site, you can create a scenario based on conversion funnels and most popular products.

- To create a realistic test, you'll want to consider the demographic of your users. Where are they coming from? What is the percentage of web vs. mobile users? How likely are they to drop off?
    - Note: We're making the assumption that you're testing the production environment using external sources of traffic, outside of your network. This helps adequately test variables that contribute to performance like DNS, CDN, and latency.
    - Marketing intelligence: For high profile events such as a Super Bowl commercial or Shark Tank airing, doing some research around historical traffic patterns from similar businesses can help inform your testing scenarios.
        - Example: According to Similar Web in 2018, stellaartois.com experienced a 40x increase from average traffic levels after its Super Bowl commercial aired, with the majority of requests coming from mobile.
- As you plan out your test, you'll want to think about the ramp up. It's best practice to increase the number of users throughout the duration of the test versus ramping up immediately to the maximum total. You'll have better insight to the level at which the environment may reach capacity or a breakdown. To illustrate, if you ramped up 5,000 concurrent users within the first minute of the test and it failed, it's difficult to determine how many users caused that failure. By contrast, if you increased the number of users by 500 every five minutes, you'll know at which level of concurrency failure happens. If performance issues surface during the middle of the test, that's also an opportunity to tune up the environment.
- After performing a stress test and optimizing both the site and infrastructure, you may be ready to run a test that most represents your upcoming event. Depending on the type of event, you may design the ramp up around the traffic patterns expected for a defined period of time. For example, if it's Black Friday, does your site experience any significant bursts of traffic? If so, at which time and level of concurrency? Design your test with these patterns in mind.

Do as much as you can before testing

If there are any known bottlenecks or areas of optimization, we highly recommend tuning up your site before a load test. That way, there's one less variable to address and your team can focus on any new problems that surface afterward. After all, these improvements are low hanging fruit.

Let's say you did make changes to the site after the test, the results of any previous load tests would essentially be invalidated costing more money to run more tests after each fix. With your business' bottom line in mind, it's best to perform these tests as effectively as possible.

What if we don't have enough time?

You realize the big event is happening in less than a week and you don't have time to load test. To effectively prepare for the event, WP Engine customers should consult with their account manager to determine best next steps. At this point, the most realistic option may be to scale up the infrastructure.

Additionally, here's a list of recommendations that could substantially improve your chances for a successful event (and normal day-to-day performance overall):

- **WPE Support:** Notify our Support team as soon as possible. This will help our Support technicians determine any optimizations that can be made to lessen the load of heavy traffic.
- **Force site to HTTPS:** It's best practice to set your entire site to HTTPS. With all URLs secure, you'll benefit from HTTP/2 which is the latest HTTP protocol. This upgrade comes with speed improvements, reduces web page latency, and improves SEO,all without changing your existing code.
- **Extend caching:** By default, Varnish caching purges every 10 minutes on our platform. Extend the cache length by using the WP Engine Advanced Cache plugin.
- **Virtual waiting room:** Determine the need for a virtual waiting room such as Queue-it. These tools allow you to effectively manage the number of concurrents by offloading excess users to a virtual queue. For ecommerce sites, if customers are willing to wait, this feature could be well worth it to prevent lost revenue.

- **MyISAM to InnoDB:** For optimal read/writes operations, we always recommend using InnoDB over MyISAM. For more details, see: wpengine.com/support/database-optimization-best-practices/
- **Autoloaded data:** By reducing the amount of data that has to be loaded on every single page request, there's less work the database has to do: wpengine.com/support/database-optimization-best-practices/#autoload
- **Enable Object Cache:** Turning this option on within the User Portal will help make your database more efficient. Complex database queries can be stored and retrieved via Memcache without the database having to do all the work.
- **CDN:** Leveraging globally dispersed edge servers to store static assets will help offload work from the origin server and improve performance overall. Example: Cloudflare
- **Disable WooCommerce Cart Fragments:** This is a quick way to reduce the number of ADMIN-AJAX requests, improve overall cacheability, and site speed of your site. Does the shopping cart total need to be updated without refreshing the page? If it's a "no," then we recommend disabling this feature: wpengine.com/support/how-to-disable-cart-fragments-for-woocommerce/
- **WooCommerce Custom Orders Table:** By default, orders are created as custom post types and stored within wp_postmeta. For every single order, WooCommerce creates over 40 separate post meta entries. As this table grows, queries can take longer to run and result in a slower site experience for customers. Good news—WooCommerce is currently working on a solution: creating separate, custom orders tables better designed for ecommerce: woocommerce.wordpress.com/2018/07/17/woocommerce-custom-product-tables-beta/
- **Redirect to cached page:** You may consider redirecting your traffic to an optimized, cacheable page, assuming it doesn't affect conversions. For example, if you're running an ad during the Super Bowl, does it make sense to link visitors directly to the home page? If the home page is more dynamic, consider redirecting/linking visitors to a cacheable landing page to help reduce the concurrency impact to the site.

Running a load test.

Load test scenario

Note: This scenario is fictional and simplified for the purpose of demonstration:

WP Engine will be airing its first ever TV commercial during the Super Bowl. Right after the commercial, we expect a spike in traffic with a max of 2,000 concurrent users on the site for at least 10 minutes. However, we're not sure if the current production environment can handle the load. In this case, we're going to run a stress test to understand where and when it'll reach capacity and breaks.

To prepare, we cloned wpengine.com to a dev site with the domain wpe.codes. From there, we migrated the new site to a dev environment that replicates production. This includes caching exclusions, redirects, and third party services including Cloudflare for DNS and CDN.

According to Bloomberg, a Super Bowl ad costs $5.3 million in 2019. In a real world scenario, peak concurrency could be over 10k users.

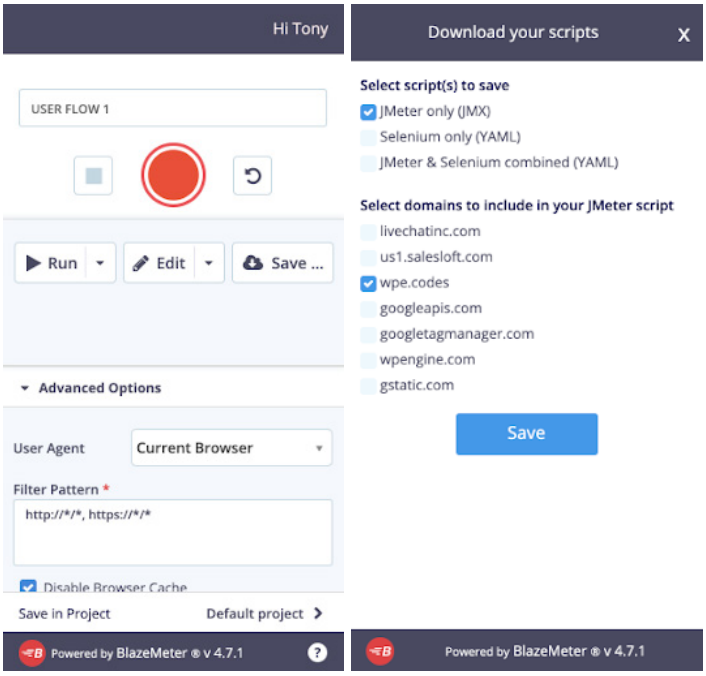
User scenarios

Based on metrics from Google Analytics and parsing our access logs, we've defined five user scenarios:

1. User visits home page → bounces immediately (40%)
2. User visits home page → navigates through several pages → drops off (30%)
3. User visits home page → navigates through several pages → visits plan page → drops off (25%)
4. User visits home page → chooses a plan → signs up (1%)
5. User visits home page → visits plan page → chooses a plan → signs up for plan (4%)

To capture this activity for our load test, we're going to use Blazemeter's Chrome Extension which allows you to record scripts in Apache JMeter and Selenium format. By using Jmeter, we can get the full picture of how much load the production environment can handle. With Selenium, we can better

understand the UI throughout the user flow. However, it's recommended to run Selenium tests carefully since they're resource intensive on the load engine. Therefore, it's better to run Selenium on functional tests—not load tests.



Depending on what systems you're testing, you may decide to include or exclude specific domains. For this example, we're only testing the dev environment with the domain: wpe.codes.

Note: The Chrome Extension is helpful for users who don't have prior scripting knowledge. Although there are many ways to create scripts, this tool provides a great basis for creating testing scenarios if you're just starting out.

For a step-by-step guide, here's Blazemeter's guide.

If you have up-to-date HAR files for other third party services, you can use Blazemeter's converter tool to convert to a JMX friendly format.

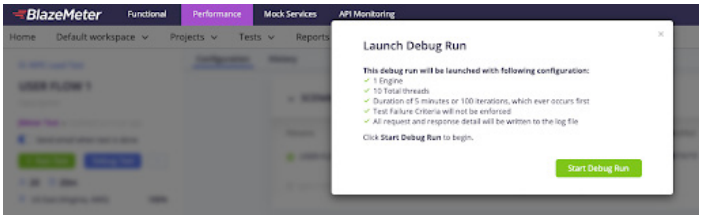
Prepare WP Engine environment before testing

To prevent issues and expedite your load testing setup, we recommend doing the following:

1. **Submit a Support request to configure the environment for load testing.** This replaces the need to whitelist multiple ranges of IPs from Blazemeter's Engines. It also prevents our platform from blocking IPs suspected of malicious activity. If you're using a third party WAF, you may have to whitelist any relevant IP ranges to ensure successful testing.
2. **Submit a Support request to disable rate-limiting for the duration of testing.** If our platform detects an excessive number of requests to the site from a single IP, the system will respond with 429 errors. For the real live event, this is not necessary, since a high volume of requests would not typically originate from the same IP.

Launch Debug Run

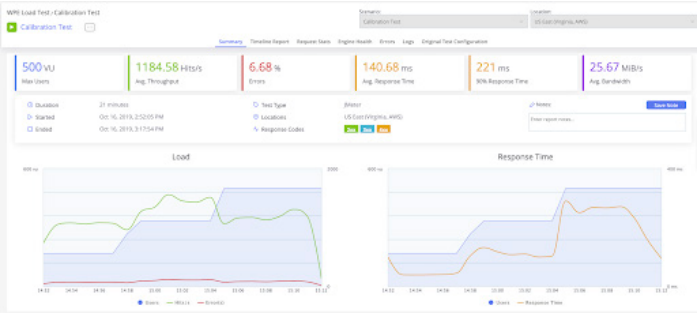
For every script created, it's best practice to run a debug test to validate the configuration. Blazemeter offers a free low-scale test in a sandbox environment with enhanced logging features. This is recommended before running a large scale test to ensure the scripts are setup properly.



Calibration tests

After creating and debugging the scripts, the next step is to determine how many users can be applied to one load engine. Just as we are testing capacity for the production environment, we need to test for the maximum number of virtual users each load engine can handle. Particularly with larger volumes of concurrents or resource intensive scripts, this helps prevent Blazemeter's testing platform from being the bottleneck.

To start, it's recommended to set the calibration test with 500 threads per engine and use only one engine. Here are the results of the first test:



After the test, you can view the "Engine Health" tab to assess the performance of the load engine.

In this case, CPU is consistently over 98% which means we'll have to decrease the maximum number of users per engine. Additionally, most of the errors were 429 responses. Since we did not disable rate-limiting on the dev environment, we will have to redo the calibration test with these adjustments.

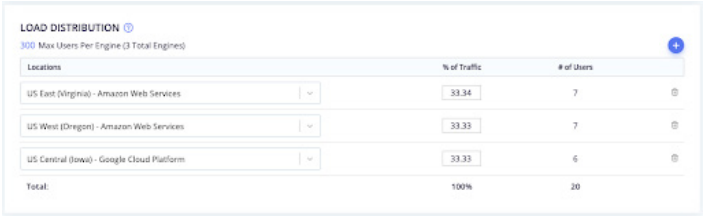
As a note, everytime you change your script, you'll need to recalibrate to understand the resource impact on the load engine. For more details, here's a guide to calibration testing.

Test configurations.

Once your scripts have been uploaded, debugged, calibrated, and tweaked as-needed, you can start setting up the configurations of each scenario.

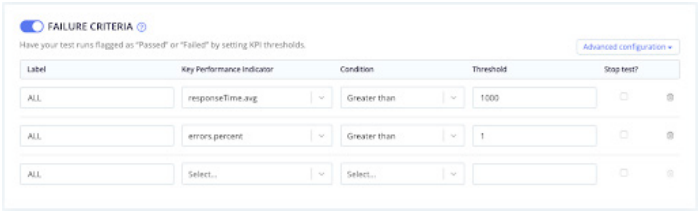
Load distribution

Since this is a nationally televised event, most of the traffic will be from the U.S. For each user flow, we have set up East, West, and Central regions for the load engines.



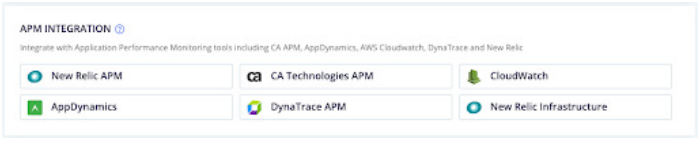
Failure criteria

Depending on acceptance and failure criteria for your business, this will vary. For this test, we have setup the following conditions:



APM integration

If you have your own New Relic APM license, you can integrate the service within Blazemeter's portal. If you're using APM directly with WP Engine, you'll need to monitor performance directly in the NR dashboard.



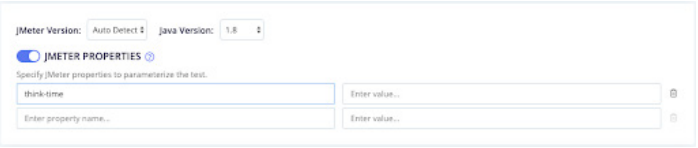
JMeter properties

Within the portal, you can set or override parameters in your scripts. You can also use the Apache Jmeter application to edit your JMX script at a more granular level.

An example of a property would be think time. Typically, users require time to consume content and decide on the next action. However, by default, JMeter scripts will send requests without pausing between each action. That's not realistic and can easily overwhelm the test environment quickly.

Fortunately, this can be prevented. By using the Blazemeter Chrome Extension, you can simulate real user behavior by adding a delay between user actions. These timer values will be automatically defined within the JMX script based on your recording.

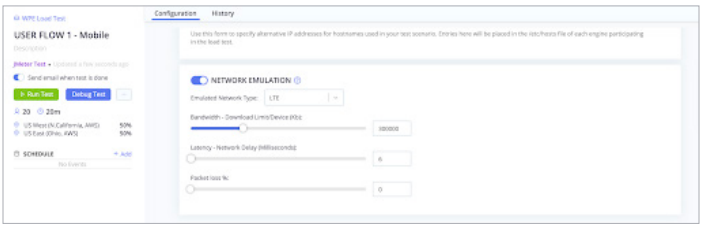
If you export a Jmeter and Selenium script into a Taurus YAML file, the timer values will be defined by the think-time variable. Assuming your scripts simulate realistic user behavior already, this section should be used only when necessary.



Emulate mobile traffic

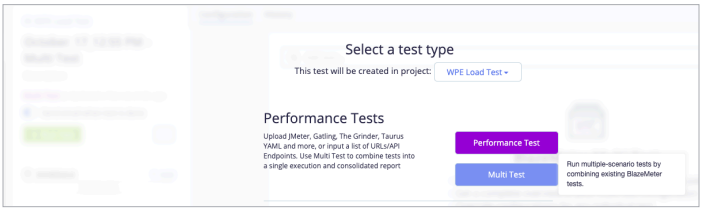
Based on internal discussions and marketing insights, we predict an increase of traffic will come from mobile devices (normally 35%). For the test scenario, traffic distribution will be 50% mobile and 50% web.

Within Blazemeter's GUI, you can duplicate tests easily to create scenarios specifically for mobile networks.

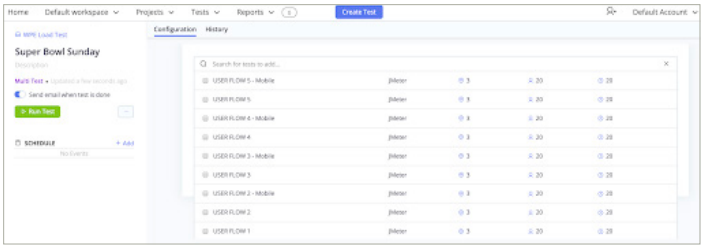


Create a multi-test

To create a test with multiple scenarios including web and mobile user activity, you can create a multi-test.

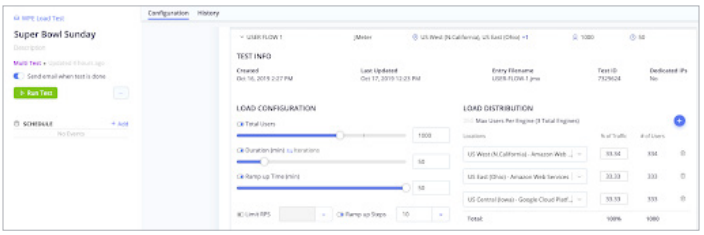


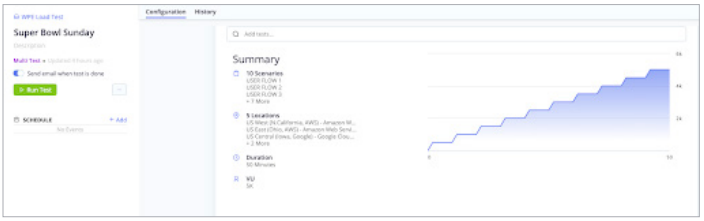
With 10 user flows total, we can run this in a single test to simulate a realistic scenario that will deliver meaningful test results.



Ramp-up configuration

The plan is to stress test the environment to it's breaking point. With this in mind, we tested for 5,000 concurrent users (2,000 is expected for event). Our team decided to ramp up by 500 users every five minutes to better understand any performance issues that might surface at each step.

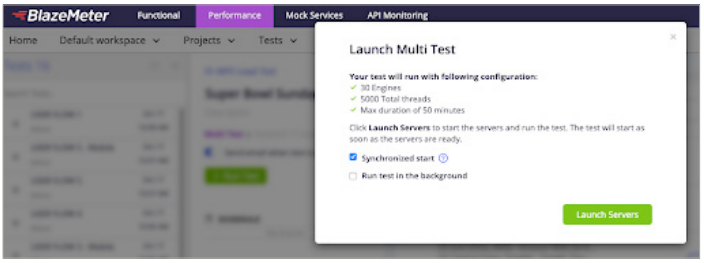




Running the test

Before launching the test, we went through a quick checklist to make sure the test ran smoothly with more accurate results:

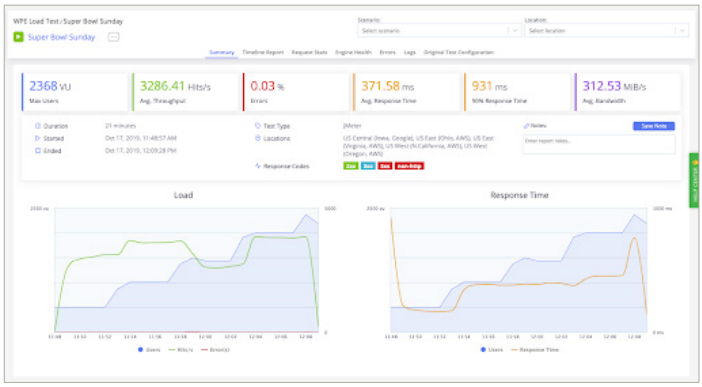
- Dev environment does not have references to the live site
- Platform configurations on prod environment are replicated to dev. Ex. Cache exclusions, page cache lengths
- Dev environment has complete stack including Nginx, Varnish, Apache, MySQL, PHP 7.3, DNS, CDN
- Infrastructure resources on dev is the same as prod
- Monitoring tools are setup for dev
- Internal comms are sent out to our teams in preparation for the influx of simulated traffic. This includes defining technical point of contact, escalation procedures, and timeframe of test.



Analyzing results

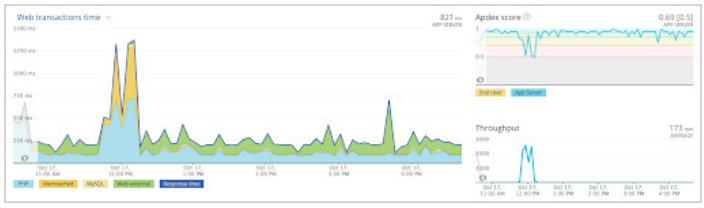
During the test, we monitored the real-time data closely and discovered the number of hits/s plateaued at 1,000 users. Around 1,400 users, we started to see 504 error codes which means requests were evicted. This can occur when the nginx queueing system is full and cannot process any additional requests. During this period, CPU utilization began to consistently reach 98%. Although there is not always a direct correlation between 504s and CPU spikes, this was a case where we needed to investigate further.

As the scenario proceeded with ramping up users, our team decided to terminate the load engines gracefully at 2,000 users. At the surface and based on the Blazemeter metrics, failure criteria were not met. However, we reached a breaking point in terms of how many requests can be supported.



Timeline report:

According to New Relic APM, web transaction time is normally at 250 ms. During the load test, web transaction time spiked up to 1,300 ms per request with most of the processing time coming from PHP and Memcached. This would be an area to dive into deeper.



Within the User Portal, you can retrieve the current Apache logs for the day. Parsing through the logs is a good start to understand which requests are most resource intensive. This could be the root cause of the performance issues at our breaking point.

```
cut -d' ' -f6,7 wpecodes.access.log | sort | uniq -c | sort -rn | head
```

Here are the top requests hitting Apache from the load test:

```
18037 "POST /integration/
17655 "POST /service/
17353 "POST /blog/wp-engine-best-customer-experience-world/
10710 "POST /plans/
7451 "POST /agency/
7239 "POST /about-us/
```

Since these pages should be cached, it's abnormal to see a significant number of POST requests. After further investigation, our development team discovered a bug within the live chat integration. This issue only became apparent during heavy load since these pages do not experience a high volume of requests normally.

With the root cause narrowed down, we plan to resolve the issue and run the stress test again with the expectation of improved cacheability and capacity.

Final thoughts.

It's important to note that load testing is an iterative process. In addition to continuous testing and optimizing, different situations call for different tests. In this particular example, a stress test was performed to understand the site's breaking point. After reviewing results and some fine-tuning, the next logical step would be to

perform a spike test to see how the production environment responds to an immediate surge in traffic. For ecommerce businesses preparing for Black Friday and Cyber Monday, you may consider following up with a soak test to understand performance over a longer period of time.

If your WordPress site has more complexity in setup, then we recommend evaluating how you're setting up the scripts. Example: For a headless WordPress site, multiple applications may be talking to WordPress at once. You'll want to design your test with your entire ecosystem of services in mind such as API calls, cron jobs, and authentication methods.

For developers who prefer to design and run test scenarios via CLI, there are many open source tools available such as Taurus, Locust, and Grinder. These are all compatible with Blazemeter's platform.

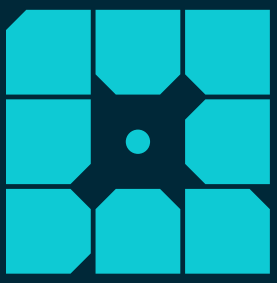
Lastly, load testing can get complicated. For example, testing WooCommerce can be a challenge. It's straightforward to test a user scenario which includes adding a product to cart and going to checkout. But unless the user completes checkout, the load test won't reveal the whole picture. A realistic scenario involves users going through the entire funnel of making an order. From the backend, orders involve a write to the database, which is critical to test. As of right now, there is no simple way of designing these types of tests. Fortunately, there is an abundance of resources available in the market. We have a few suggestions:

- Leverage professional services for their expertise and industry knowledge:
    - blazerunner.io
    - support.loadimpact.com/3.0/how-to-tutorials/testing-e-commerce-checkout-process/
- Experiment with 3rd party open-source solutions: github.com/Coded-Commerce-LLC/WooCommerce-Load-Test
- Develop load testing knowledge to execute more complex tests: blazemeter.com/jmeter-tutorial/

About the author.

Tony Le

Tony Le is a WP Engine Sr. Solutions Engineer, AWS Solutions Architect Associate professional, and New Relic Performance Pro certified. He loves discovering interesting, innovative technology, and learning how it can help transform the digital experience, all while creating ways for customers to win online.



About WP Engine.

WP Engine is the world's leading WordPress digital experience platform that gives companies of all sizes the agility, performance, intelligence, and integrations they need to drive their business forward faster. WP Engine's combination of tech innovation and an award-winning team of WordPress experts are trusted by over 120,000 companies across 150 countries to provide counsel and support, helping brands create world-class digital experiences. Founded in 2010, WP Engine is headquartered in Austin, Texas, and has offices in Brisbane, Australia; Limerick, Ireland; London, England; Omaha, Nebraska; San Antonio, Texas and San Francisco, California.
www.wpengine.com

wpengine.com