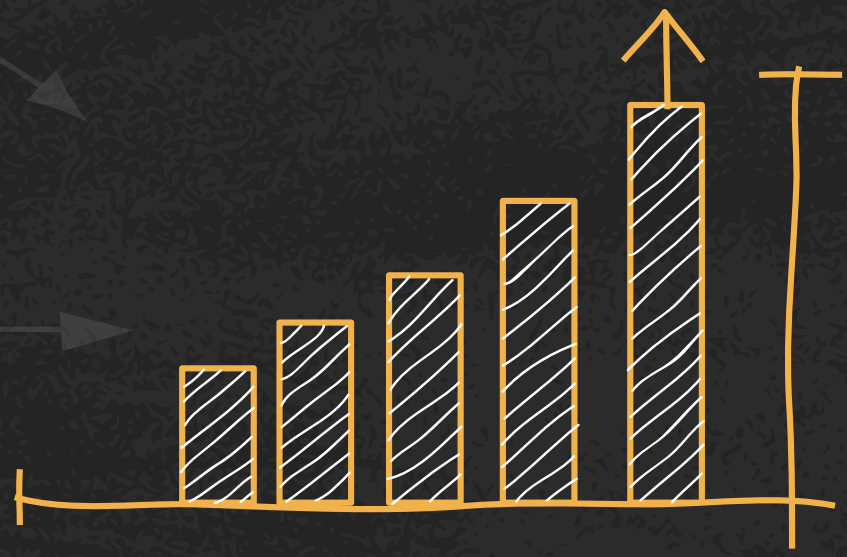


API LOAD TESTING 101

A Beginner's Guide to Best Practices, Key Metrics, and Impactful Results

LoadUI Pro

LoadUI Pro

Load UI Pro is hands down the easiest way to run a quick API load test, either against a single web service endpoint or based off of an existing functional API test created in SoapUI Pro

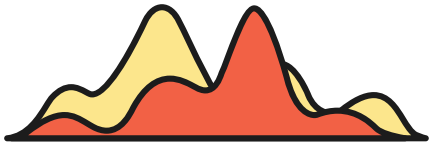
LEARN MORE ABOUT **LOADUI PRO**

Content

This book is filled with tips for load testing and performance testing, as SmartBear's Ready! API suite of applications is the world's most comprehensive toolset for testing APIs. Its purpose is to serve as a fundamentals guide for beginners and experts alike.

We aim to take you on a journey from functional testing to load testing — and how they are intrinsically linked — the can't-be-understated importance of load testing, its requirements, and then we'll dive into the nitty gritty: running exploratory load tests, uncovering traffic bottlenecks, and all the metrics surrounding this key practice. We'll finish by helping you analyze the results. Along the way we'll explain how LoadUI Pro can not only help you get started with these load testing best practices, but also automate a reassuringly repeatable workflow!
In the end, we hope you'll get out of this more questions than answers — because that's what great software testing is all about: formulating the right questions about your product and how your users interact with it, and then, whenever possible, developing **automation testing**, so you have more time to ask more questions... and then the cycle continues.

So, good luck on your API load testing journey. We hope this guide makes it anything but stressful for you and your team!



What is load testing vs. stress testing vs. performance testing?

These are terms that, for better or worse, are often used interchangeably. If nothing else, load, stress and performance testing are interrelated and all equally important to the success of your business. Often you will see load testing and stress testing falling under the umbrella term performance testing, which encompasses testing the performance level of the various aspects of any system.

Load Testing Basics

Load testing specifically provides us insight into the behavior of our systems, comparing the current state of usage to what would happen if we hit specific loads of users, simultaneously made calls or processed transactions.

Load testing is important to your business model because it allows you to monitor your system's response times for each of the transactions during a set time period. It provides your business development and project management teams with crucial data on usage, while raising developer and QA attention to any potential problems with the software. It helps you find things that are wrong now before your users do, usually at inconvenient times when you're busiest, both in terms of traffic and staff demand. It also allows you to find bottlenecks before they occur and choke your business.

As we'll talk about later in this guide to load testing, it doesn't just involve your breaking point, but rather it gives you insightful load test reports and metrics, including predicted and actual traffic patterns, CPU and memory, and other ways to not only monitor one-time situations but usage overtime. More specifically, load testing usually includes your end user's response time, so you can see how the user experience could potentially change if behavior and usage does.

Finally, load testing allows you to predict how a planned release will work (or not) before an update goes live.

Don't get us wrong — depending on how you feel about fortune telling — we wouldn't call load testing a crystal ball, but rather something with a high probability of success, simulating how your tool will behave out in the wild.

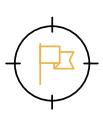
Load Testing Basics

One way of describing it is extreme load testing. Stress testing is a kind of performance testing that happens when you push your app, API or software to the upper limits of its capacity. It works to reveal how your system will hold up under a sudden surge of demand when your current number of users goes beyond the maximum levels you can support. Or, if you don't know what that maximum level is, stress testing is a good way of establishing where it falls.

KINDS OF PERFORMANCE TESTING

Spike testing. What happens when you hit that max stress level quite suddenly?

Configuration testing. This type of testing helps you find any changes to the pieces of your system that affect behavior or performance.



Endurance testing. This monitors continuous load, red-flagging any slow leaks that may be slowing you down or wasting resources.

Isolation testing Isolation testing is used to try to zero in on a specific problem in hopes of finding its cause and fixing it.

Comparative testing As its name suggests, it involves comparing the performance of two or more systems, both to find anomalies and sometimes to make a competitive decision. It also facilitates shared learning and cross-company intelligence.

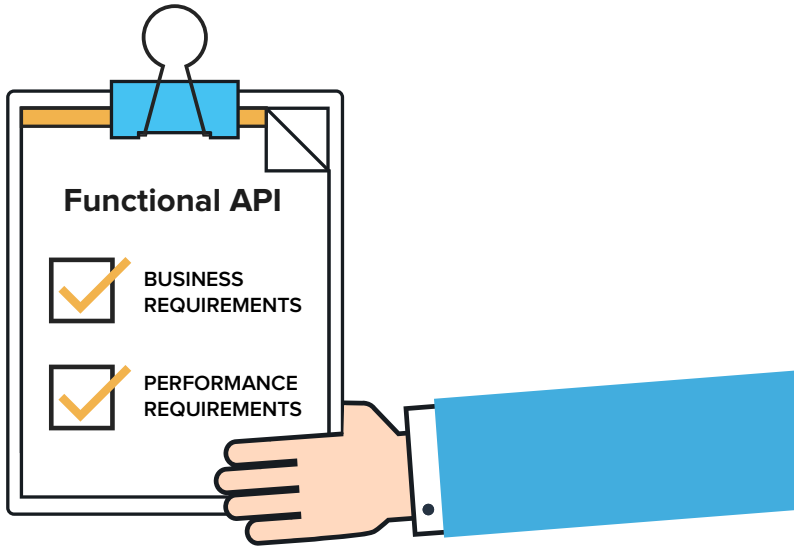
Custom testing Certain industries and certain kinds of customers require certain kinds of testing that need to be ongoing for the success of the brand.

In the world of **continuous deployment and integration** or CICD, API testing that you are continually integrating is another important aspect of overall performance testing. After all, if your app works, but it can't successfully pull or push data, it's not really useful is it? And when you have a set of business processes and actions linked together by APIs, it's essential that you make sure that not only your API is functional, but that nothing in that

workflow tumbles. This is also sometimes called **interoperability testing** because it sees how code works across a variety of platforms, focusing first on the mission-critical aspects of those workflows.

We found the topic of **"moving from functional testing to load testing"** to be somewhat of a silly question. ***When we asked quality analysts and testers, they more and more seem to look at load testing as one aspect of*** **functional testing**, or, as we already talked about, load testing is part of performance testing which is part of overall functional testing. After all, can something really be functioning if it only functions at a lower level? Sure a house can handle sunshine and rain, but the foundation is only really tested in a hurricane — and we'd certainly expect it to (hopefully) keep standing through that hell or high water too. So why should your software, website, or app be any different?

Functional testing focuses on how code stands up to business requirements, mainly focusing on risk criteria. It also typically is done in a prioritized fashion where testers are organizing from most to least important functions. And, of course, almost always, load testing should be one of your top priorities — unless you aren't worried about having a lot of customers actively engaged with your product.



If performance testing is an umbrella, functional testing is more of a tent, which is why some tech journalists have tried to argue that it "is not easily turned into a repeatable process." However, with the evolution of cloud computing combined with **automation testing**, this has dramatically changed. So much of functional testing — and, in particular, load testing — can be automated in a way that allows you to continuously ask those questions that are essential to your business' success. Specifically with API testing, the cloud has allowed for an in-depth process of business-oriented functional testing.

Why is load testing so important?

If the benefits of the various kinds of functional testing laid out above didn't persuade you of its utter importance, we're not sure what might. And yet, so often, code is tossed over into the QA and testing departments without much thought into its importance.

To put it bluntly, if you don't do performance testing, your app, software or API — and by extension your business — is likely to crash and burn. That's why it's called performance testing: it keeps track of how your product is performing and behaving. Of course, if you only have a handful of customers, you can probably let load testing go, but that's not very ambitious then, is it? And then what if one of those precious few customers you can't afford to lose suddenly accelerates its usage? Then, you won't only crash that one customer but the whole lot of them.

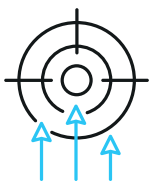
Basically, your business is too important not to load test. No matter what you predict for your traffic and usage rates, this is a definite hope for the best, prepare for the worst situation.

You could even say that load testing and performance testing act as an extension of your customer service and your overall design. This kind of testing should be done from the start of your product — to determine viability, interest, and to build reliable performance into the design and structure of the system — until the day you're ready to retire it. Especially important is automation testing for that time in between, so your code can adapt and flow and your resources can contract and expand to meet client and server demand.

As one developer wrote, "No matter how good application features might be, users will lose interest if it is performing poorly with crashes or delayed load time. The application should be able to withstand peak load, perform under lower network bandwidth and should not affect battery performance."

At the end of the day, load testing is an important business move. Cigniti, a QA services firm, lists five economic advantages to automated load testing in the cloud:

1. Automated Load Testing is Cost Effective. Now that it's in the cloud and run with tools like LoadUI Pro, you save time and money by creating tests that are repeatable and that don't involve expensive hardware requirements.



2. Automated Load Testing is More Efficient. Now that testing automation is in the cloud, it has more flexible resource allocation. As you are connecting different tools via different APIs, getting a singular view of these different pieces means even more cost cutting.

3. Automated Load Testing is More Collaborative. You get all the benefits of an in-person office, with a singular view of all your testing automation, but without the need to be in the same place. Cloud-based testing automation also means that you can choose when you are running it — Continuous testing? Or perhaps only at night when you can wake up to reports? Or perhaps for load testing it's best during your peak usage.

4. Automated Load Testing is Fast. High productivity in shorter test cycles, quick setup and deployment. While you will want to continue to write your own tests, testing automation gets you up and running almost instantly.

5. Automated Load Testing is for Anyone. On smaller teams or on more agile teams, you can't afford a dedicated QA or tester, so each person tests his or her own work. This type of continuous and collaborative testing means not only that anyone can potentially test what he or she needs to, but the result is a more transparent business that allows everyone to participate in this critical business matter.

Warning: Failure to load test may be your downfall

There are countless reasons why load testing is so essential to your business. But perhaps the recent examples of failure to adequately load test are the most persuasive.

How about last Autumn when the New York Stock Exchange went down for over three hours? With Wall Street only relatively recently automated and without significant backup and recovery systems, the market saw a huge drop not just in trade activity but in value and consumer confidence because they hadn't adequately prepared for a surge in stock movement.

Or take for example, one of the world's largest airlines, United, had to ground flights around the globe for about an hour, delaying thousands of passengers on an airline already notorious for having the worst on-time average. One United pilot said "It's like someone pulled the plug on our computers - it's embarrassing. I apologize." Then all the boarding pass information of those grounded passengers had to be re-input by hand. All because United hadn't load tested for their busiest month of the year.

Even Amazon — yes the company that probably is backing your servers — had an epic #PrimeDayFail, when, on its "Black Friday in July," many customers experienced site outages. Now, as the original dogfooders, some say this was done on purpose, perhaps putting the employees of this now known questionable culture on edge, but wouldn't it be better to just have automated and simulated that load testing ahead?

And this wasn't the first time Amazon lost out because they weren't ready for a rush. In the 2013 holiday season, the world's biggest retailer went down for 40 minutes setting it back $5 million.

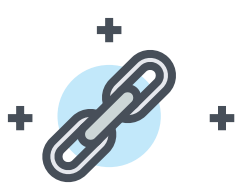
How do you calculate the cost of downtime?

Downtime Losses =

Minutes of Downtime x Average Revenue per Minute

Yes, the bigger they are, the harder they fall — or really, experience a small stumble in the spotlight. But as users become increasingly demanding and sensitive to performance, there's no doubt that if you're not a Fortune 100 company or an entire marketplace, you are even more susceptible to failures and losing companies.

Each year, downtime equals billions of dollars in losses.



You want to do whatever you can to avoid this:

Service Unavailable

HTTP Error 503. The service is unavailable.

Nobody is patient enough these days. They will immediately Google for alternatives, **probably your competitor**.

But this is all enough on the importance of load testing — because you get it's essentialness, right? Now it's time to get into how to make sure you aren't only load testing but loading test well and covering all your requirements.

What are your load testing requirements?

The best way to start any testing process is by figuring out your own requirements and by talking to your current customers to figure out theirs — after all, who would know better? For APIs, this may be all endpoints, a specific endpoint or a subset you are looking to improve the performance of. It's about knowing what your specific test is for and why.

Like with all good testing, you can start to ask questions, like: How many users do we have at peak time now? How many could we have? How many users would we have in an ideal world? These numbers are great to begin setting your load testing parameters.

Always remember this rule: The more realistic your load testing is to your real-life workload, the more useful your load test will be.

This means that it's not terribly useful to keep your load tests at a thousand requests per second if you're only at 100. But, on the other hand, if you are only testing at 100, you aren't preparing for the future and the reasonable scaling you may face soon. This is why, before you begin creating a load test, **you must first research your current API usage**.

Start by answering the following questions:

- What is your average number of requests per second?
- What is your peak throughput? What is the most traffic you could have over a certain period of time?
- Which, if any, endpoints get the most traffic?
- Is your traffic generated by all users or can you pinpoint a few hyper users?

According to 3scale by RedHat's Victor Delgado, you should look for patterns and usage statistics that can offer insights into:

- Repetitive load generation
- Simulated traffic patterns
- Real traffic scenarios

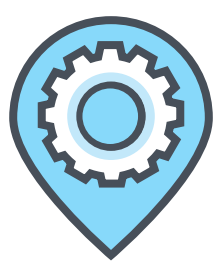
He goes onto explain that you want to start by asking simple questions for simple answers. When in doubt, these are the most realistic and accurate answers.

“Running your firsts tests with repetitive load generation against your API endpoints will be a great way to validate that your load testing environment is stable. But most importantly, it will also let you find the absolute maximum throughput of your API and therefore establish an upper bound of the performance that you’ll be able to achieve,” says Delgado.

This is a great starting off point to understand your realistic traffic.

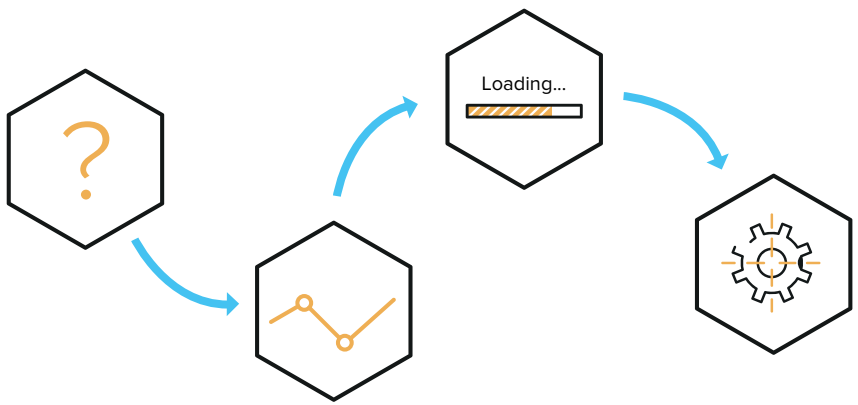
He goes onto suggest that you can also kick off your API load testing with your API production logs as a way to replay your most realistic use cases.

Between these two examples, you should be able to recognize not only what kind of traffic is typical, but also which endpoints are creating this traffic.



The basics of load testing

Before we go deeper, let's review your load testing basics.



Step 1: Ask the right questions.

To reiterate (and then again), all great testing is about asking questions and then asking them again. Here are some questions you should be asking about API load testing or any kind of performance testing for that matter:

- What kind of things are important for you to test for?
- What are your expectations?
- How are you setting your performance goals? What are they?
- How do your real conditions differ from your testing conditions? (To go right into this, scroll down to the section labeled, "Four ways to load test your API")

Step 2: Set a baseline, or the number of requests your API can handle per second without timing out or returning errors.

Like any science experiment, you want everything in your testing to remain constant while you only change one variable at a time. A good place to start your experimentation is with the CPU usage, since your users are likely sensitive to slow responses and it's a likely culprit for overloading.

This baseline can often be referred to as your one-x or 1x (just like algebra class back in the day!).

Step 3: Kick it up a couple notches.

Continuing to draw on our Algebra 101 memories, now we are talking 2x, 3x, and so on, multiples of your one-x.

You're not trying to break everything at once. Start with that baseline you established and then slowly crank up the power and number of calls — because, unless you're already having poor responses and systems breakdowns, your API should be able to handle more. When you hit around 99 percent **latency distribution** or wait time, you have hit your maximum acceptable delay in response.

Distributed load testing involves using multiple systems or cloud options to put stress on your system, mimicking an acceleration of your different simultaneous users performing API calls or website calls. This is a good way to find out both latency and request response time, as well as the time it takes to process it all.

As this Quora respondent probably explains it best:

> *Request response time =*
> *Processing time + Latency*

"Suppose a request is sent from the browser to a server. There is a certain time duration before it reaches the server and actual processing is started by the application server based on that request (say an API call). [T]hat time is latency or network latency. There could be numerous reasons for latency but they are mainly related to network configurations and geo-locations," says QA Manager Anand Singh.

If you are performing a load test on an already existing website, finding out the number of virtual users to throw at it is quite simpler. Your existing traffic is your baseline and then you can start by doubling, tripling and quadrupling what you're throwing at it. But, if you are working with an API, you don't usually care how many users are simultaneously reaching out to you. Instead, you care about any latency in requests per second.

Step 4: Discover the origins of it all.

Now that you know how to count your load weaknesses, it's time to figure out where they are coming from. This is where we get into how to actually calculate these numbers in a way that doesn't break your whole system.

Four ways to load test your API

The Smartbear blog has already covered this in another article, but here we will dive deeper into the advantages and disadvantages of these four pathways to a tested load.

IF YOU BUILD IT, THEY WILL COME: SET UP A FULL TESTING ENVIRONMENT

Remember when we talked about the possibility that Amazon was actually dogfooding during their #PrimeDay-Fail? If that is true, it's not recommended. Whenever performing testing, you want to set up a testing, pre-production or sandboxed environment. You wouldn't want to risk things collapsing in a live production environment, would you?

Sometimes called **clone load testing**, this is done by making a full copy of everything you have running on and integrating with your code. This works well if you don't integrate with a lot of other apps or if you do and can handle the pay-per-call (PPC) cost of pulling from and pushing to them.

This is your most realistic testing environment, without risking reality, but, besides that pesky PPC, it has some other downsides. This can take up a lot of CPU usage on its own because you need to reload each time to send and retrieve stateful data. Also, depending on your customers and your compliance restrictions, it can risk sharing sensitive data and privacy restrictions.

FAKE IT TIL YOU MAKE IT: MOCK YOUR API

This is common especially when your API isn't released yet or when you are going to release some updates that might lead to a massive spike in traffic or a slowed latency that you want to prepare for.

Mocking up your API is really only good in the earliest stages of development, when you are just trying to check basic stability. Since you don't have real data yet, they are rather like the first mock-up of a website — a rough sketch that most likely won't represent reality in the end. **API mocking** is an interesting way to just play around with your API early on but it's simply not the best way to test your API's performance.

GAMBLING YOUR API: TESTING IT LIVE

This is an approach where teams — who are often trying to cut corners with testing in general — will save time by testing right in their production environments. Yes it does save time and, in the short-term, money, but it's risky! The best way to do this is if you actually warn your users ahead that you will be out-of-use for a couple hours, but then you are going to be risking current customers.

Plus, more worrisome, what if you corrupt or crash your production server? Expose customer data? Risk the security or stability of your whole API? So while this is still a pretty popular path — that does have very realistic results — it's not one we'd recommend.

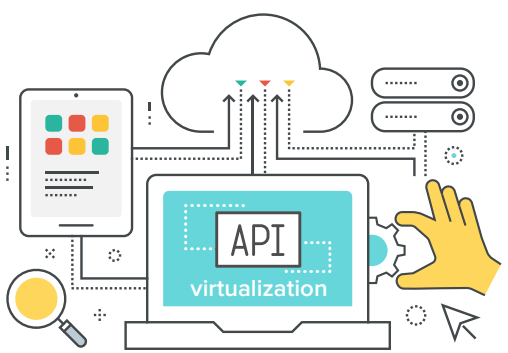
TALK ABOUT AUGMENTED REALITY: TRY API VIRTUALIZATION

API virtualization takes things to a whole new level past cloning. This preferred option allows you to test your actual API in an environment where you're able to control each load variable. A virtual API is like a mock, but on steroids. This allows you to isolate each variable in your experiment, including:

- Environmental loads on network bandwidth
- Server connections
- Database connections
- Simultaneous users

You get the results that you can get from clone load testing, but without the ache of having to set up a whole duplicate production environment. You can create your own requests or clone and run the actual requests and responses your API receives every day.

With API virtualization, you can even control the speed of the requests. Heck, you can even simulate the other external APIs you are linking to so you don't have to pay-per-call for them either.



Spike testing helps you check how the tested API With this opportunity, it takes the least amount of work to safely isolate and test different aspects of your API.

And, of course, like all proper load testing, if done correctly, API virtualization allows you to catch and fix issues before it all hits the fan when your customers catch it first!

Taking it past response time: Key performance indicators for load testing

Sure, it's pretty easy to break your application, website or API under an excessive load. But figuring out why and how it broken isn't so simple. In this section, we highlight the areas of load testing metrics you should look out for to help you get to know your API, its limitations, and your users better.

Before we break down the kinds of areas you should be considering — because it's not just your API and its metrics that will be able to measure performance — think about the following, as each ultimately affects your user experience:

RESPONSE KPIs

- **Average response time** - time to first byte or last byte
- **Peak response time** - tells your longest cycle
- **Error rate** - percentage of problems compared to all requests

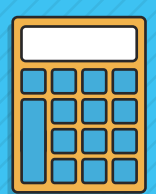
VOLUME MEASUREMENTS

- **Concurrent users** - many virtual users are active at any given time
- **Requests per second** - how many requests for HTML pages, CSS stylesheets, XML documents, JavaScript libraries, images, Flash/multimedia files, etc. to the server
- **Throughput** - often bandwidth consumed, but, in general, the maximum rate at which something can be processed

VIRTUAL USER CALCULATION

- **Virtual Users (VUs)** - A certain number of users simultaneously accessing your system or a certain number of users accessing from different browsers
- **Session length** - A group of interactions that took place on your website at a certain time, like how long did someone spend on your app or website, including jumping around to different pages
- **Peak-hour pageviews** - Your base can come from your Web analytics tool (like Google Analytics). Also find your peak traffic time (perhaps holiday rush?), then increase with percentages of expected traffic growth

- **Concurrent user** - Runs through a transaction from start to finish then repeats
- **Single user** - Only one transaction completed
- How many actual users do you predict will access your system at once?
- How many VUs do you need?
- How many rows of data do you need?
- How much bandwidth do you need?



CHECK OUT THIS VIRTUAL USER CALCULATOR FOR LOAD TESTING

Now that you know these key load testing terms, we break down how these will play out for you and your load testing results and where they are testing which parts. With the help of DevOps and Web Performance Tester Andreas Grabner, you'll learn about some of the most important load testing KPIs.

LOAD TESTING METRIC #1: WEB SERVER METRICS

Web server metrics help you find errors in your API deployment, so you can scale and augment as needed:

- **Busy and idle threads** - Do you need more Web servers? More worker threads? Do you have an application performance hotspots slowing you down?
- **Throughput** - How many transactions per minute can your API handle? When is it time to scale to more Web servers?
- **Bandwidth requirements** - Is your network your bottleneck? Or is there content that is pulling it down that you can offload?

LOAD TESTING METRIC #2: APP SERVER METRICS

Whether your application server is Java, PHP, .NET or something else, here's where you can try to find deployment or config concerns:

- **Load distribution** - How many transactions are handled by each engine? Do you have load balance? Or do you need more application servers?

- **CPU usage hotspots** - How much CPU usage do you need for each load? Can you fix programming to lower CPU or do you simply need more?
- **Memory problems** - Is there a memory leak?
- **Worker threads** - Are they correctly configured? Are there Web server modules that block these threads?

LOAD TESTING METRIC #3: HOST HEALTH METRICS

Sometimes it's not your API's fault at all. These Web and application servers run on hosts. Grabner offers us these host tests:

- **CPU, memory, disk, input/output** - Problems with network interfaces? Are we exhausting resources?
- **Key processes** - Which processes are running on our host? Should we be taking some resources off or should we be redistributing them to other virtual or physical machines?

LOAD TESTING METRIC #4: APP METRICS

If you have either created an application or if your APIs are connecting to them, you'd be remiss to not investigate how each part of your apps handles loads and scales:

- **Time spent in logic layer** - Which layer slows down with an increased load? Which layers scale or don't scale well?
- **Number of calls in logic layer** - How often are you calling internal Web services? How often are you calling into your critical APIs? Are they your own APIs or others'?

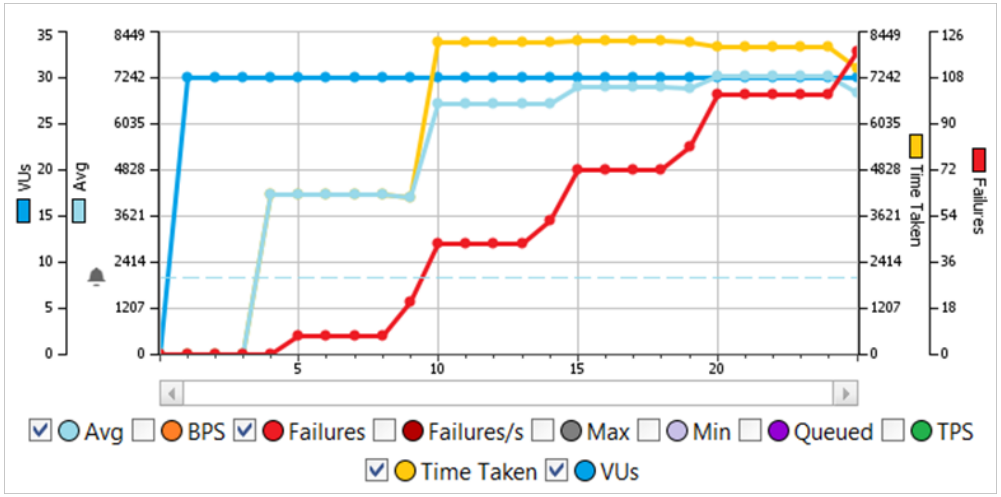
LOAD TESTING METRIC #5: API METRICS

Your API performance affects mobile and Web apps, which means increasingly impatient users will quickly uninstall or Google for your competitor.

As we say at Smartbear, your service level agreement (SLA) is a promise that you cannot afford to break. API load testing metrics get into specific kinds of throughputs:

- **Transactions per second (TPS)** - in any number of transactions presented, how many are able to go through and how many have to queue?
- **Bits per second (BPS)** - bytes divided by time passed

- **Queued (arrival)** - who is being left behind?



Understanding your load test reports

Now that you know what you are looking for and how to get to it, it's time to take a step back to look at the big picture of the results of your load testing. Doing so will ensure that you are able to act on them and aren't just another crash test dummy, but instead are prepared for whatever load comes your way.

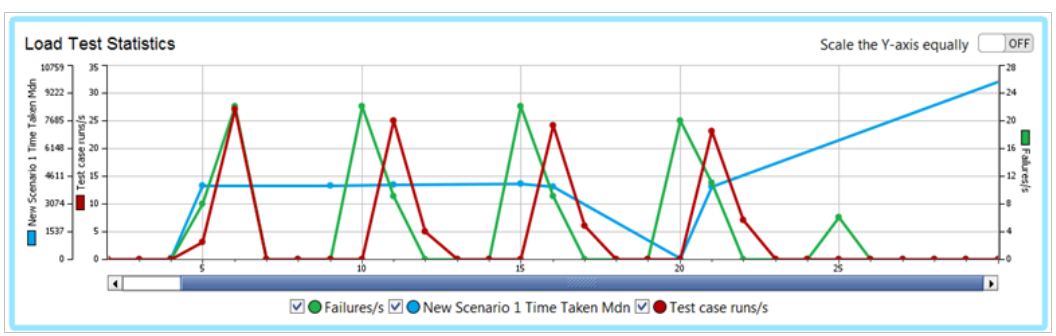
INTERPRETING LOAD TESTING RESULTS

You have performed your first load tests and you have a lot of data returned. That data needs to be analyzed in order to become useful information. If you are using an API load testing tool like LoadUI Pro, you will be able to generate a lot of this information programmatically and automatically. However, you need to work then to integrate those graphs in with more meaningful, actionable information.

With so much of testing automated by the right tools, this interpretation and explanation is where performance testers earn their pay.

PRESENTING YOUR RESULTS

So you, mighty tester, have created a hypothesis, drawn tentative conclusions, and repeated the process to verify them. Now it's time to share it all with your team. Load testing isn't the most riveting conference room topic. You not only need to gather your performance testing results and understand them, but often you need to be able to present them to people who are perhaps less technical than yourself. If nothing else, you will have to present them to your project manager that will need persuasion as to why he or she needs to allocate resources — time and money — toward fixing these issues, increasing load capacity and decreasing latency.



TIPS FOR EFFECTIVELY PRESENTING YOUR RESULTS

Visualize your data. Spending all our days staring at screens, it's definitely the images — not the words — that will help people see where spikes in traffic lead to de-

lays in output. Try simple graphs, but be clear in explaining what each access represents.

Keep It Simple, Stupid. The KISS Rule of journalism applies here. Slide decks are boring. Create a few slides — try to keep it down to four or five — that summarize and visualize the basics.

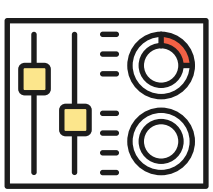
Don't forget the "why" Simon Sinek's Golden Circle applies here too. Make sure you offer the compelling reasons why you should improve your performance. Make sure to talk about its importance to UX and customer retention.

It's all about the money. Have access to impact on revenue data? Lead with that. The cost of customer downtime or API call latency can be super persuasive.

Don't forget your SLA. Your service level agreement is your promise and even legal contract with your customers. Are you meeting it? If you aren't, that's definitely worth a compelling slide.

Focus on your action plan. What have you and your team already done to overcome your load unbalance? What do you plan to do? How can you reallocate resources better? How much time will it take? When should you be able to deliver by?

When all is done, share your story! It's a great use case for others to mimic about your load testing experience, as well as illustrates your commitment to improving your customer service.



Utilizing the Right Tools

As you get started with your load testing, SmartBear has the tools you need to ensure that your APIs perform flawlessly under various traffic conditions. LoadUI Pro is the industry's leading API load testing tool that is great for beginners, because it's scriptless and allows for easy reuse of your functional API tests from SoapUI Pro.

LoadUI Pro allows you to quickly get started and:

- **Save time & resources** by building load tests from pre-configured templates in just a few clicks
- **Create real-life traffic patterns** from servers 'on premise' or in the cloud
- **Understand server performance** by visualizing the effects of load on your servers with real-time monitoring
- **Quickly analyze results** by collecting advanced performance metrics for your load test runs and benchmarking them against past tests
- **Reuse your existing functional test cases** from SoapUI Pro for increased efficiency

Can’t get enough?

Dive into these further load testing resources:

- What is Load Testing? [Article]
- Ensuring API Speed & Performance [eBook]
- Reusing Your Existing Functional Tests as Load Tests [Webinar]
- API Functional Testing & Load Testing [Webinar]
- Getting More from Your Functional Tests by Reusing Test Scripts [eBook]
- Virtual Calculator for Load Testing [Calculator]
- Analyzing the Results of Your Load Testing Reports [Article]

SMARTBEAR

Over 4 million software professionals and 25,000 organizations across 194 countries use SmartBear tool

4M+ users

25K+ organizations

194 countries

See Some Succesful Customers >>

API READINESS

Functional testing through performance monitoring

SEE API READINESS PRODUCTS

TESTING

Functional testing, performance testing and test management

SEE TESTING PRODUCTS

PERFORMANCE MONITORING

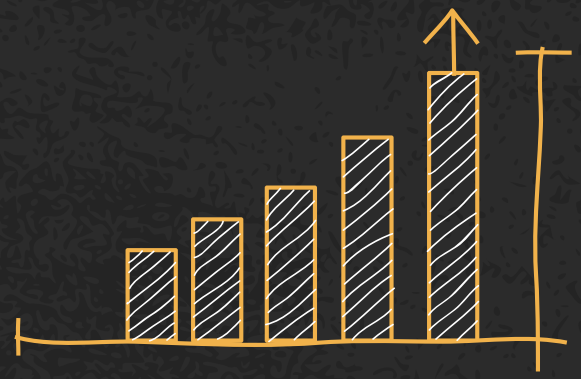
Synthetic monitoring for API, web, mobile, SaaS, and Infrastructure

SEE MONITORING PRODUCTS

CODE COLLABORATION

Peer code and documentation review

SEE COLLABORATION PRODUCTS



LoadUI Pro