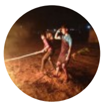
Introduction to Load Testing



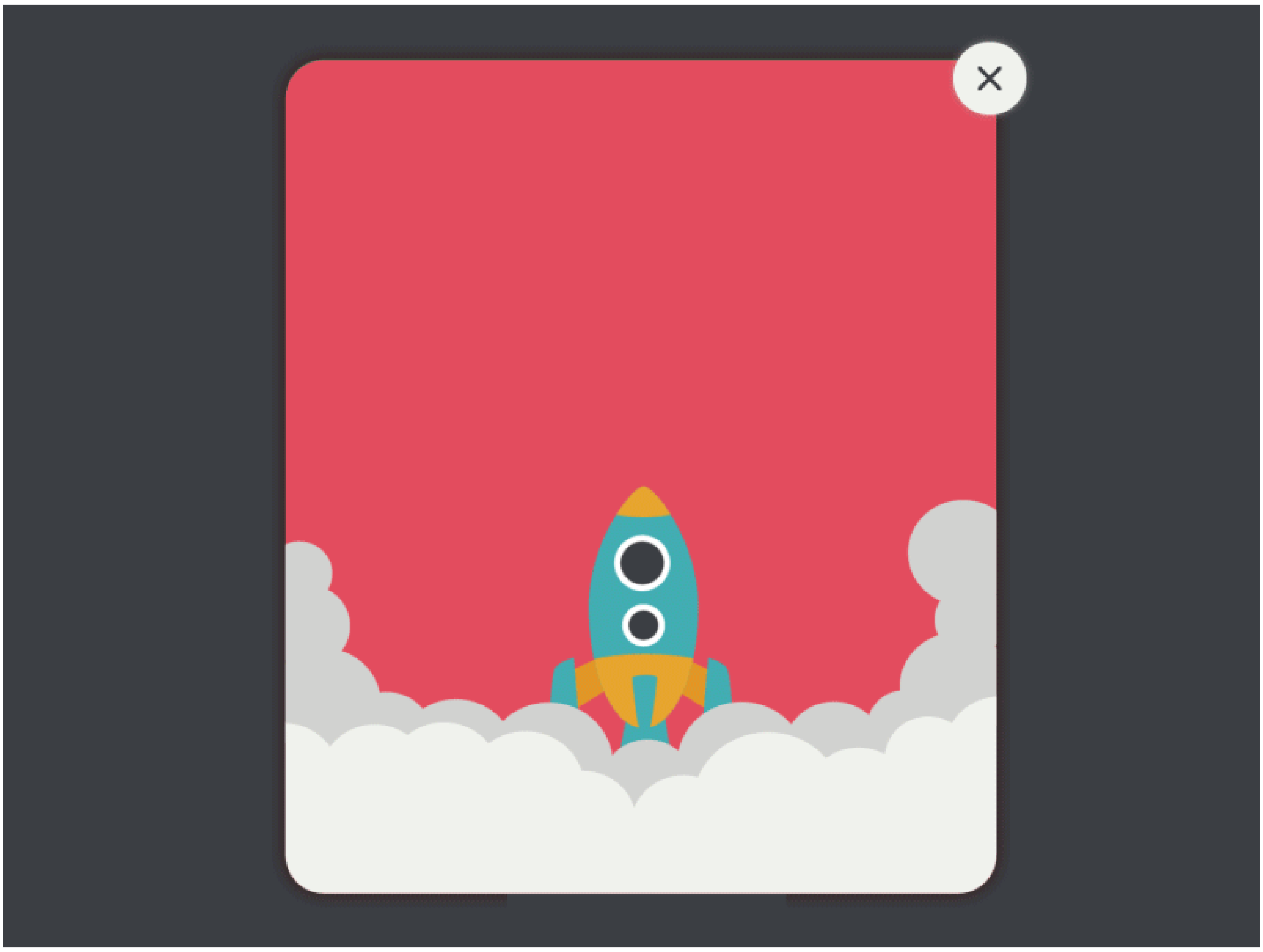
Rohit Khatana
Apr 27, 2018 · 6 min read

So, Load testing What, Why?

The first thought which comes to our mind that why do we need to do load testing, as we know our code working fine on our dev machine. Or it is working fine since the first launch :)

But hey!! let me tell you something it's not like that, everything has a limit. And knowing your limit is always helpful. Sometimes you are not ready to face the problem which can occur when the load increases. So Whenever a developer writes the code/design the software, he must know that when will his code/architecture breaks.

Ok, Now we agree, we need to understand the application limit. But how can we identify the pain points in our application?



So I decided to write about it because when I started doing this, it was very hard to start, there are plenty of load testing framework or etc, but which to decide, And after deciding the framework what to do now, how I start now. So I accumulated the ideas and knowledge which I have in this article. So that anyone can start doing load testing in an easy way.

. . .

Before doing load testing you should have some metrics in your mind which you want to achieve for the application.

We should try to achieve metrics for 1 instance/machine(app server):

Throughput: 100 RPS(requests per second) or 6000 RPM(request per minute)

Latency: < 100ms(time taken by the server to send back the result), but for best performance, it should be less than 50ms

CPU: < 50%

Memory: < 50%

Again these numbers are just for giving you some context. They should be less than whatever I have specified above. Except for throughput :)

Deciding the Load testing framework/tool

The basic and essential part of load testing is deciding the right tool for you, which can be used. They are plenty of loads testing tool out there like locust.io, AB(apache benchmark), Jmeter, Gatlin, Vegeta etc

If you just want to fire the load on your system easily, you can use AB. But if you want to generate traffic like real-time application will have, custom weight, scenario etc, use locust, Vegeta, Jmeter etc. My personal choice is the locust, as it is written in python with easy installation, and you can write your scenarios easily.

Setup a server and app level monitoring service

Before you start doing your load testing, you must set up a good server monitoring service like **new relic**, **Nagios** etc. Without this, you will not be able to find the CPUs, memory usages etc. And also setup app monitoring, this will help you to find out the bottlenecks in your application.

Find your load pattern

You should have a lot of end-points in your backend. But every endpoint will have its own characteristic. Means, Endpoints can be hit at the different rate like lower, average, high. Then there will be some type of flow for the using the endpoints from your client. And at last every endpoint will have different payloads too. Knowing these factors will help to design the test cases. An easy way to start is to find out the critical endpoints. And then do the load test on them. Later combine all the endpoints for generating production like traffic. Because sometimes single load test will give the good response, but when we combine all these services, they behave differently.

Generate your load data

This is an essential part of getting the best results. Because when you do your load testing without generating the load/seed data, your test case will not give you the exact output, because sometimes you missed the indexes or unnecessary have used indexes. These things are hard to find out in a small dataset. So I will suggest for generating the data use your system APIs only so that you can also identify the problem with them too. But sometimes you do not have that much writes in your application, in that case, you can insert the data directly into the DB, and can test the APIs too. It depends on the use case. But I will suggest inserting at least 1 million records in your tables/collections for getting the good insights for your platform.

Tune your OS and TCP

The very basic steps would be tuning your operating system. But not necessarily the first step, because when you start doing the load test you will get this feedback in the first go itself. The very basic error which people get in there first to go is **too many files open.** It is because of, OS has a limit on the number of the active file descriptor. Because every TCP connection opens a socket for which a socket descriptor is opened which is ultimately a file descriptor. So your server will start rejecting the new TCP connection after the limit is exhausted. We can set any number between 1–65535, depending on our use case.

You can verify your limit by this command

```
$ ulimit -n # file descriptors
2560
$ ulimit -u # processes / threads
709
```

Also by default, your OS is not configured for the server use case. If you're not using server ready OSs. So you may need to tune your TCP config also.

Use ***this*** blog for reading more about the TCP tuning.

Tune your application

All this tuning part is optional for getting started with your load testing, as after doing the load testing you will have your results, which will force you to fix/tune your application. But as a good practice before doing load testing, read about production level configuration for a framework like the number of workers, connection pool size, memory limit, concurrent requests limit etc. After reading all these things, this will help you to tune your application according to your need.

For example number of workers should follow this formula: 2n+1, where n is number of CPUs.

Enable profiling for your DBs

It is recommended to turn on the DB profiling for your application, it will help you to find out the queries which are taking too much time. Like in MongoDB be default: MongoDB logs the queries which take time more than 100ms. But you should tune the profiling for your DB to a lower limit. For mongo, profile read ***this*** article.

Deciding the hardware

I think deciding the hardware should be the last step after you have tuned your software part like OS, application, DB etc. Because if you do not tune your software part, you might start using the bigger hardware, which might not be required. Again for finding out the best hardware for you is start with the minimum hardware like 512 MB ram, 1 core CPUs, and minimal hard disk space. But it’s very slow process, so you can read the documentation of the applications which you are using, mostly all have specified the hardware config. But it is a good way of evaluating your application limit with respect to hardware. So that you can tune application on your own. For JVM based system it is recommended to start with 2GB ram.

So deciding the hardware is like start with minimal hardware, and run your load test, follow the feedback, upgrade the hardware accordingly.

It’s like **RED-GREEN**-REFACTOR rule when you do TDD.

And for calculating the disk space for your application is very easy, just find out the size of 1 record per table, then multiply it with the number of records you are expecting in next 1 year. And add at least 20% growth rate also. And then you can know the disk size for your DB.

Create load Server

And the last thing should create a load server who will fire the load on your services. It can be the simple machine on which os tuning and load framework are set up. And try it to be on the same network (VPC). As we do not want to add a network latency.

Simulate different network speed

When we have done with our load testing. We should also evaluate the different network band because our customers will have a different kind of network.

For JMeter, you can use this link to simulate different network speed.

https://www.blazemeter.com/blog/how-simulate-different-network-speeds-your-jmeter-load-test

Run your test for longer time

Also run your load test for at least 1 hour. Because the performance of system can be reduce over the time. As there can be memory leak orthreads are releasing the resource fast enough etc, as over the period there are chances that the result can vary. So it's a good practice to run your load test for a longer period.

Conclusion:

I hope I explained everything clearly enough for you to understand. If you have any questions, feel free to ask. You can find me *on twitter*.

Make sure you click on green heart below and follow me for more stories about technology :)

Load Testing Python DevOps Performance Engineering