Load Test Report

Date: 7/27/2016

Test from: virginia

Query URL: http://2016pressable50.reviewsignal.com/

Started at: Wed Jul 27 2016, 04:44:10 -04:00

Finished at: Wed Jul 27 2016, 04:45:10 -04:00

Test link: https://www.blitz.io/to/#/play

Analysis

This rush generated 77,850 successful hits in 60 seconds and we transferred 1.38 GB of data in and out of your app. The average hit rate of 1,298/second translates to about 112,104,000 hits/day.

The average response time was 132 ms.

You've got bigger problems, though: 0.02% of the users during this rush experienced timeouts or errors!

Response Times
Fastest: 131 ms
Slowest: 135 ms
Average: 132 ms

Test Configuration
Region: virginia
Duration: 60 seconds
Load: 1-3000 users

Other Stats
Avg. Hits: 1,298/second
Transferred: 11.32 MB
Received: 1,401.79 MB

Hits

This rush generated 77,850 successful hits. The number of hits includes all the responses listed below. For example, if you only want HTTP 200 OK responses to count as Hits, then you can specify --status 200 in your rush.

<table>
<thead>
<tr>
<th>Code</th>
<th>Type</th>
<th>Description</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>HTTP</td>
<td>OK</td>
<td>77850</td>
</tr>
</tbody>
</table>

HTTP 200 OK 100% (77850)

Errors

The first error happened at 10 seconds into the test when the number of concurrent users was at 496. Errors are usually caused by resource exhaustion issues, like running out of file descriptors or the connection pool size being too small (for SQL databases).

<table>
<thead>
<tr>
<th>Code</th>
<th>Type</th>
<th>Description</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>TCP</td>
<td>Connection reset</td>
<td>8</td>
</tr>
<tr>
<td>23</td>
<td>TCP</td>
<td>Connection timeout</td>
<td>3</td>
</tr>
</tbody>
</table>

Connection reset 73% (8)

Timeouts

The first timeout happened at 52.5 seconds into the test when the number of concurrent users was at 2625. Looks like you've been rushing with a timeout of 1000 ms. Timeouts tend to increase with concurrency if you have lock contention of sorts. You might want to think about in-memory caching using redis, memcached or varnish to return stale data for a period of time and asynchronously refresh this data.
The max response was **134 ms @ 3000 users**

The max hit rate was **2,555 hits per second**