# Stress Test Results Report

|  |  |
| --- | --- |
| Tested Software | Sprint 03 Project: Flask ML API |
| **Tester** | Leandro Ordoñez |
| Purpose | |
| Stress-test the performance of the Machine Learning Flask API endpoints that classify images using a Convolutional Neural Network | |
| Testing objectives | |
| * Performance validation of API’s response rates * Evaluate scalability needs * Measure the stability and reliability of the API under heavy load * Test how the API performs under pressure | |
| Performance Metrics | |
| * # of requests: total number of requests made to the API * # of fails: total number of requests that failed * Median, percentiles, average, min and max: response time in milliseconds * RPS: requests per second * Current failures/s: total of failures per second | |
| Test Environment | |
| Hardware | Intel Core i9, 10850K. 3.60 GHz, 10 cores. 32 GB RAM. |
| Software | Windows 10 64 bit. Docker 26.1.4 |
| Test methodology | |
| I used Locust 2.25 to stress test two endpoints of the API: index and predict. For each test, I tested with different number of users and rate of requests. I also tested increasing the instance count of the ML service.  Before executing the tests, I installed locust with the following command:  pip install locust  To launch the tests, I ran:  locust  from the /stress\_test folder. Locust is running in this endpoint: http://localhost:8089  The test cases defined in the file locustfile.py  Two test cases are defined:   * test\_index: Performs a GET request on the index API * test\_predict: Performs a POST request on the /predict endpoint. It sends a default image as payload.   Result data is collected from the locust user interface and raw data was downloaded as CSV | |
| Test results | |
| I tested the service with 1, 10, 50, 100 and 500 concurrent users, at a spawn rate of 1, 10, 50, 100 and 100 users/second respectively. | |
| Test results with one user | |
| |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | Type | Name | # Req | # Fails | Median (ms) | Avg (ms) | Min (ms) | Max (ms) | Avg Size | Current RPS | Fail/s | | GET | / | 20 | 0 | 3.01 | 5.34 | 3 | 11 | 553 | 0.1 | 0 | | POST | /predict | 23 | 0 | 110 | 134.35 | 106 | 715 | 87 | 0.2 | 0 | | Aggr |  | 43 | 0 | 110 | 74.35 | 3 | 715 | 303 | 0.3 | 0 | | |
|  | |
|  | |
| * No failures occurred * Median time for the index is 3 ms. Median time for /predict endpoint is 110 ms * Total ratio:   + 100.0% APIUser   + 50.0% testIndex / 50.0% testPredict | |
| Test results with 10 users | |
| |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | Type | Name | # Req | # Fails | Median (ms) | Avg (ms) | Min (ms) | Max (ms) | Avg Size | Current RPS | Fail/s | | GET | / | 138 | 0 | 3 | 3.68 | 2 | 12 | 553 | 2.1 | 0 | | POST | /predict | 134 | 0 | 110 | 127.03 | 106 | 717 | 87 | 1 | 0 | | Aggr |  | 272 | 0 | 11 | 64.44 | 2 | 717 | 323.4 | 3.1 | 0 | | |
|  | |
|  | |
| * No failures occurred * Median time for the index is 3 ms. Median time for /predict endpoint is 110 ms * Total ratio:   + 100.0% APIUser   + 50.0% testIndex / 50.0% testPredict | |
| Test results with 100 users | |
| |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | Type | Name | # Req | # Fails | Median (ms) | Avg (ms) | Min (ms) | Max (ms) | Avg Size | Current RPS | Fail/s | | GET | / | 923 | 0 | 2200 | 2285.1 | 65 | 4782 | 553 | 7.7 | 0 | | POST | /predict | 926 | 0 | 3100 | 3199.7 | 161 | 5691 | 87 | 8.7 | 0 | | Aggr |  | 1849 | 0 | 2800 | 2743.1 | 65 | 5691 | 319 | 16.4 | 0 | | |
|  | |
|  | |
| * No failures occurred * Median time for the index is 2200 ms. Median time for /predict endpoint is 3100 ms * Total ratio:   + 100.0% APIUser   + 50.0% testIndex / 50.0% testPredict | |
| Test results with 500 users | |
| |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | Type | Name | # Req | # Fails | Median (ms) | Avg (ms) | Min (ms) | Max (ms) | Avg Size | Current RPS | Fail/s | | GET | / | 703 | 0 | 27000 | 22259 | 64 | 29152 | 553 | 8.8 | 0 | | POST | /predict | 794 | 0 | 27000 | 23323 | 161 | 30188 | 87 | 8.6 | 0 | | Aggr |  | 1497 | 0 | 27000 | 22823 | 64 | 30188 | 305.8 | 17.4 | 0 | | |
|  | |
|  | |
| * No failures occurred * Median time for the index is 27000 ms. Median time for /predict endpoint is 27000 ms * Total ratio:   + 100.0% APIUser   + 50.0% testIndex / 50.0% testPredict | |
| Conclusions | |
| With this hardware and docker setup, the service did not produce any failures even when stressed. However, response times raised to between 2 and 3 seconds for 100 simultaneous users and up to 27 seconds for 500 simultaneous users. Taking into consideration that with 1 and 10 simultaneous users, the median response times were between 3 and 110 milliseconds, the response times rise steeply with 100 simultaneous users and more.  Therefore, I recommend to increase the number of instances of the service and API containers when the user count starts to raise beyond 100. If three seconds’ time to make a prediction is not an acceptable response time during a peek load, the number of instances should be increased for even less simultaneous users. | |

## Increasing the number of instances

|  |
| --- |
| Purpose |
| Stress-test the same model increasing the number of instances of the model |
| Testing objectives |
| * Increase the number of instances of the model container * Stress-test the system to validate the throughput under heavy loads * Measure the performance of the API under heavy load when increasing the number of instances of the model |
| Test methodology |
| The test methodology is the same as the previous one, except that the number of instances was increased with the following command:  docker compose up --build --scale model=5 -d  Now, there are five instances of the model container, as can be seen with this command:  Docker ps    Now, the stress test is repeated to see if the response time decreases |
| Test results |
| I tested the service with 500 concurrent users, at a spawn rate of 100 users/second. |
| Test results with 500 users |
| |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | Type | Name | # Req | # Fails | Median (ms) | Avg (ms) | Min (ms) | Max (ms) | Avg Size | Current RPS | Fail/s | | GET | / | 1563 | 0 | 6100 | 5805 | 50 | 7095 | 548 | 26.5 | 0 | | POST | /predict | 1604 | 0 | 6300 | 7100 | 81 | 7371 | 75 | 28 | 0 | | Aggr |  | 3167 | 0 | 6200 | 5930 | 76 | 7371 | 308 | 54.5 | 0 | |
|  |
|  |
| * No failures occurred * Median time for the index is 6.1 s. Median time for /predict endpoint is 6.3 s * Total ratio:   + 100.0% APIUser   + 50.0% testIndex / 50.0% testPredict |
| Conclusions |
| Increasing the number of instances of the Model container has a positive effect over the response times, which drop from a median of 27 seconds to 6 seconds (78% of response time improvement). The requests per second increase from 17.4 to 54.5 (more than 300% improvement).  **As a result, the throughput of the system is greatly improved.** |