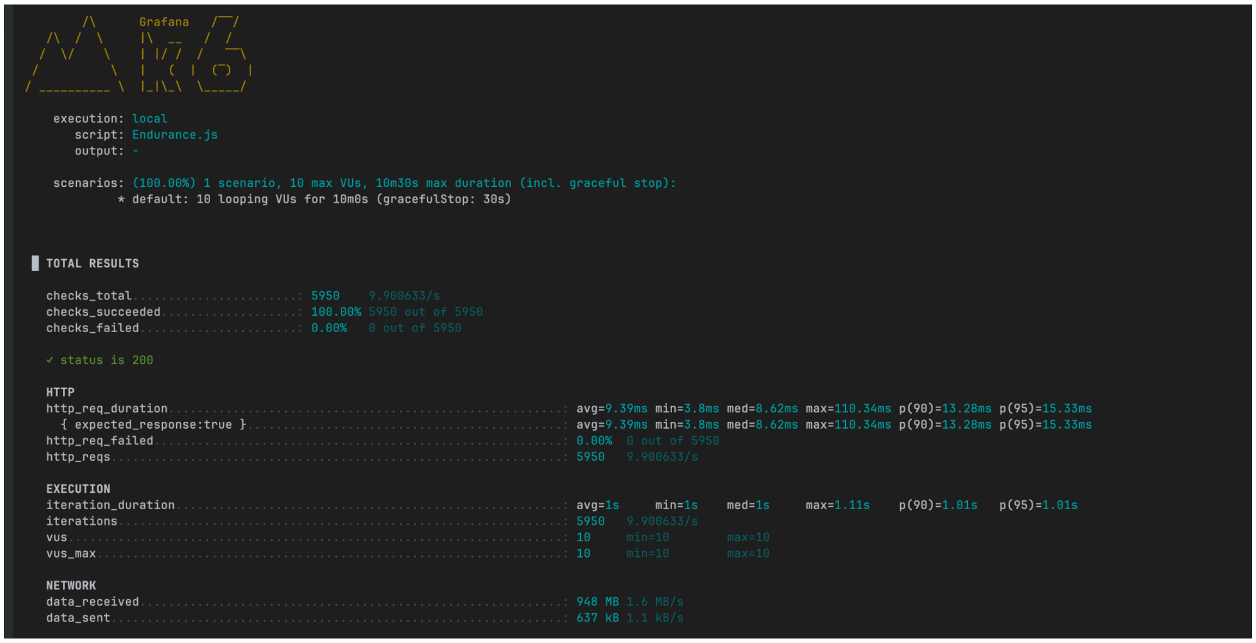


```
         /\      Grafana   /‾‾/
    /\  /  \     |\  __   /  /
   /  \/    \    | |/ /  /   ‾‾\
  /          \   |   (  |  (‾)  |
 / __________ \  |_|\_\  \_____/

     execution: local
        script: Endurance.js
        output: -

     scenarios: (100.00%) 1 scenario, 10 max VUs, 10m30s max duration (incl. graceful stop):
              * default: 10 looping VUs for 10m0s (gracefulStop: 30s)



  █ TOTAL RESULTS

    checks_total.......................: 5950    9.900633/s
    checks_succeeded...................: 100.00% 5950 out of 5950
    checks_failed......................: 0.00%   0 out of 5950

    ✓ status is 200

    HTTP
    http_req_duration.......................................................: avg=9.39ms min=3.8ms med=8.62ms max=110.34ms p(90)=13.28ms p(95)=15.33ms
      { expected_response:true }............................................: avg=9.39ms min=3.8ms med=8.62ms max=110.34ms p(90)=13.28ms p(95)=15.33ms
    http_req_failed.........................................................: 0.00%  0 out of 5950
    http_reqs...............................................................: 5950   9.900633/s

    EXECUTION
    iteration_duration......................................................: avg=1s     min=1s     med=1s     max=1.11s    p(90)=1.01s   p(95)=1.01s
    iterations..............................................................: 5950   9.900633/s
    vus.....................................................................: 10     min=10        max=10
    vus_max.................................................................: 10     min=10        max=10

    NETWORK
    data_received...........................................................: 948 MB 1.6 MB/s
    data_sent...............................................................: 637 kB 1.1 kB/s
```

Performance Testing Report

1. Introduction

The purpose of this endurance test is to evaluate the performance of the API under sustained load conditions over an extended period. The goal is to identify any potential performance issues that may arise during prolonged usage.

2. Test Environment

Tool Used: k6

Number of Virtual Users (VUs): 10

Test Duration: 10 minutes

Main Script: Endurance.js

3. Test Scenarios

The endurance test was conducted to evaluate the performance of the following scenario:

Fetching Pets: Continuous GET requests were made to retrieve the list of available pets.

4. Results

4.1 Summary of Checks

Total Checks Performed: 5950

Successful Checks: 5950 (100.00%)

Failed Checks: 0 (0.00%)

4.2 Specific Checks

Status Code 200:

All requests returned a successful status code.

4.3 HTTP Request Metrics

Average Request Duration: 9.39 ms

Minimum Request Duration: 3.8 ms

Median Request Duration: 8.62 ms

Maximum Request Duration: 110.34 ms

90th Percentile: 13.28 ms

95th Percentile: 15.33 ms

HTTP Requests Failed: 0.00% (0 out of 5950)

4.4 Execution Metrics

Average Iteration Duration: 1 s

Total Iterations Completed: 5950

Virtual Users (VUs):

Minimum: 10

Maximum: 10

4.5 Network Metrics

Data Received: 948 MB (1.6 MB/s)

Data Sent: 637 kB (1.1 kB/s)

5. Analysis

Success Rate: The success rate of 100% indicates that all requests were processed successfully without any errors, demonstrating the stability of the API under sustained load.

Request Duration: The average request duration of 9.39 ms is well within acceptable limits, with a maximum duration of 110.34 ms. This suggests that the API can handle requests efficiently even under continuous load.

Network Performance: The data transfer rates indicate that the API is capable of handling significant amounts of data without performance degradation.

6. Recommendations

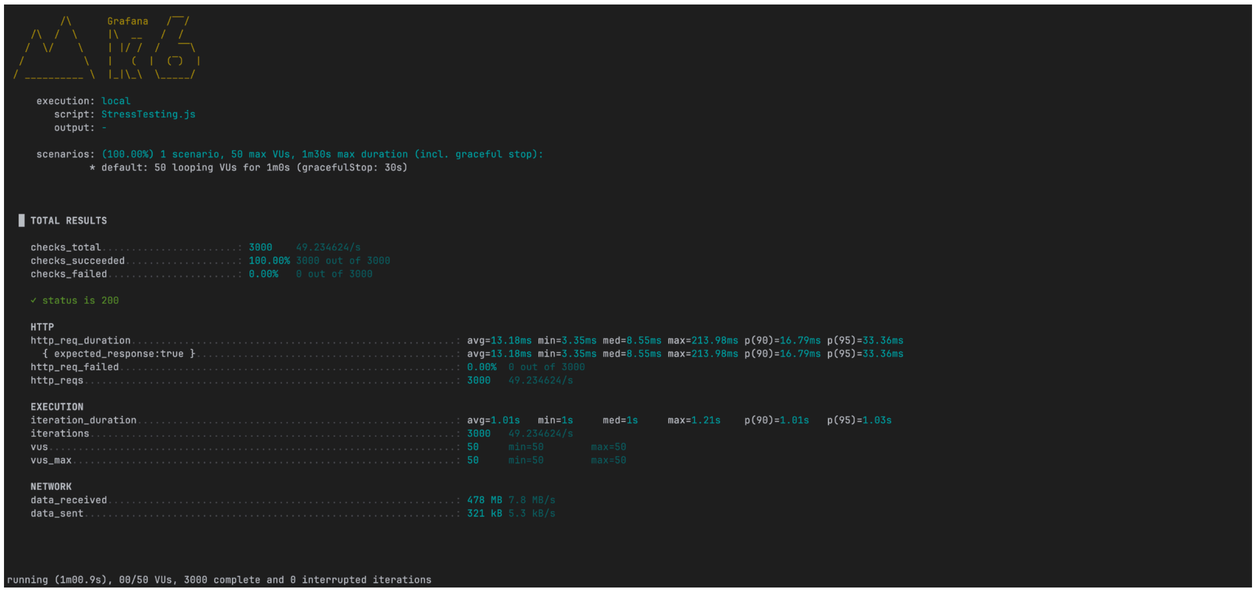
Continuous Monitoring: Implement continuous monitoring to ensure that the API maintains performance levels in production over time.

Load Testing with Increased VUs: Consider conducting additional tests with a higher number of virtual users to evaluate how the API performs under heavier loads.

Stress Testing: Perform stress tests to identify the breaking point of the API and understand how it behaves under extreme conditions.

Documentation of Results: Keep a record of these results for future reference and to compare with subsequent tests to identify trends and improvements.

Review Server Configuration: Ensure that the server configuration is optimized for handling sustained loads, including database connections and resource allocation.



```
         /\      Grafana   /‾‾/
    /\  /  \     |\  __   /  /
   /  \/    \    | |/ /  /   ‾‾\
  /          \   |   (  |  (‾)  |
 / __________ \  |_|\_\  \_____/

     execution: local
        script: StressTesting.js
        output: -

     scenarios: (100.00%) 1 scenario, 50 max VUs, 1m30s max duration (incl. graceful stop):
              * default: 50 looping VUs for 1m0s (gracefulStop: 30s)



  █ TOTAL RESULTS

    checks_total.......................: 3000    49.234624/s
    checks_succeeded...................: 100.00% 3000 out of 3000
    checks_failed......................: 0.00%   0 out of 3000

    ✓ status is 200

    HTTP
    http_req_duration.......................................................: avg=13.18ms min=3.35ms med=8.55ms max=213.98ms p(90)=16.79ms p(95)=33.36ms
      { expected_response:true }............................................: avg=13.18ms min=3.35ms med=8.55ms max=213.98ms p(90)=16.79ms p(95)=33.36ms
    http_req_failed.........................................................: 0.00%  0 out of 3000
    http_reqs...............................................................: 3000   49.234624/s

    EXECUTION
    iteration_duration......................................................: avg=1.01s   min=1s     med=1s     max=1.21s    p(90)=1.01s   p(95)=1.03s
    iterations..............................................................: 3000   49.234624/s
    vus.....................................................................: 50     min=50        max=50
    vus_max.................................................................: 50     min=50        max=50

    NETWORK
    data_received...........................................................: 478 MB 7.8 MB/s
    data_sent...............................................................: 321 kB 5.3 kB/s




running (1m00.9s), 00/50 VUs, 3000 complete and 0 interrupted iterations
```

Performance Testing Report

1. Introduction

The purpose of this stress test is to evaluate the performance of the API under extreme load conditions. The goal is to identify any potential performance issues that may arise when the API is subjected to a high number of concurrent requests.

2. Test Environment

Tool Used: k6

Number of Virtual Users (VUs): 50

Test Duration: 1 minute

Main Script: StressTesting.js

3. Test Scenarios

The stress test was conducted to evaluate the performance of the following scenario:

Fetching Pets: Continuous GET requests were made to retrieve the list of available pets.

4. Results

4.1 Summary of Checks

Total Checks Performed: 3000

Successful Checks: 3000 (100.00%)

Failed Checks: 0 (0.00%)

4.2 Specific Checks

Status Code 200:

All requests returned a successful status code.

4.3 HTTP Request Metrics

Average Request Duration: 13.18 ms

Minimum Request Duration: 3.35 ms

Median Request Duration: 8.55 ms

Maximum Request Duration: 213.98 ms

90th Percentile: 16.79 ms

95th Percentile: 33.36 ms

HTTP Requests Failed: 0.00% (0 out of 3000)

4.4 Execution Metrics
Average Iteration Duration: 1.01 s

Total Iterations Completed: 3000

Virtual Users (VUs):

Minimum: 50

Maximum: 50

4.5 Network Metrics
Data Received: 478 MB (7.8 MB/s)

Data Sent: 321 kB (5.3 kB/s)

5. Analysis
Success Rate: The success rate of 100% indicates that all requests were processed successfully without any errors, demonstrating the stability of the API under high load conditions.

Request Duration: The average request duration of 13.18 ms is acceptable, with a maximum duration of 213.98 ms. While the average is low, the maximum duration suggests that there may be occasional spikes in response time that should be monitored.

Network Performance: The data transfer rates indicate that the API is capable of handling significant amounts of data efficiently.

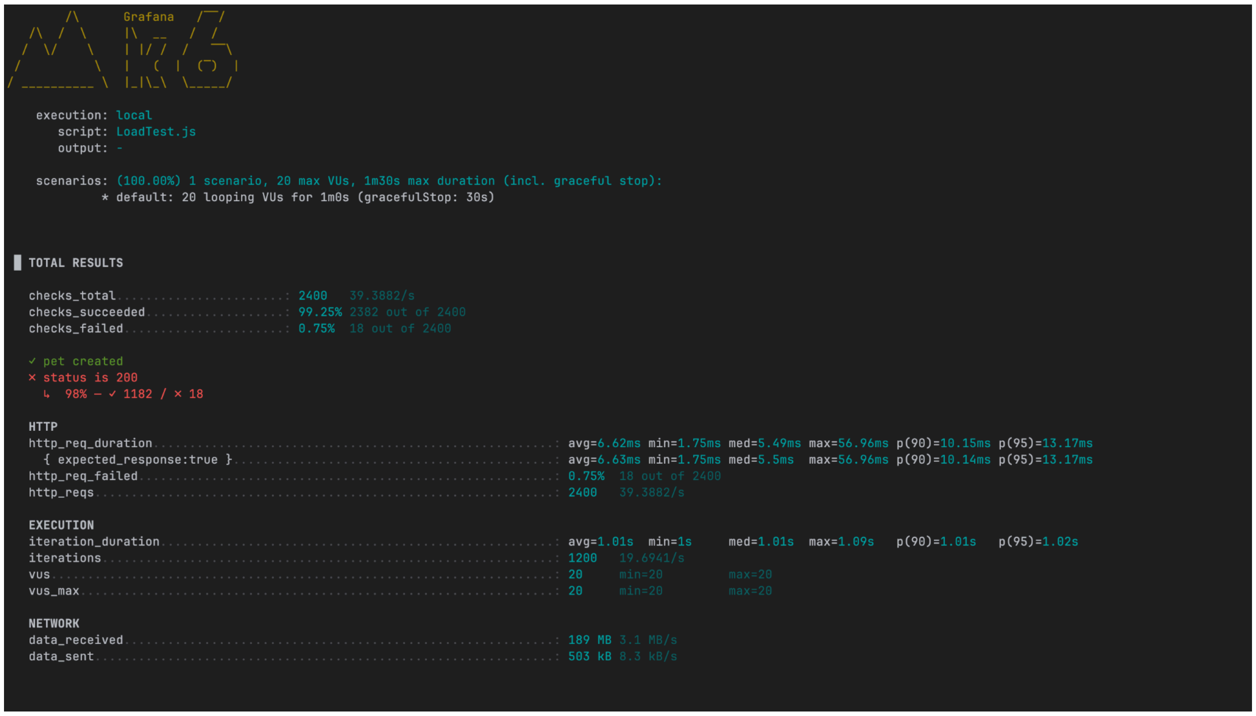
6. Recommendations
Continuous Monitoring: Implement continuous monitoring to ensure that the API maintains performance levels in production, especially under high load.

Further Stress Testing: Consider conducting additional stress tests with even higher numbers of virtual users to identify the breaking point of the API.

Performance Optimization: Investigate the occasional spikes in request duration to identify potential bottlenecks in the API or backend services.

Documentation of Results: Keep a record of these results for future reference and to compare with subsequent tests to identify trends and performance improvements.

Review Server Configuration: Ensure that the server configuration is optimized for handling high loads, including database connections and resource allocation.



```
        /\      Grafana   /‾‾/
   /\  /  \     |\  __   /  /
  /  \/    \    | |/ /  /   ‾‾\
 /          \   |   (  |  (‾)  |
/ __________ \  |_|\_\  \_____/

    execution: local
       script: LoadTest.js
       output: -

    scenarios: (100.00%) 1 scenario, 20 max VUs, 1m30s max duration (incl. graceful stop):
             * default: 20 looping VUs for 1m0s (gracefulStop: 30s)



█ TOTAL RESULTS

  checks_total.......................: 2400   39.3882/s
  checks_succeeded...................: 99.25% 2382 out of 2400
  checks_failed......................: 0.75%  18 out of 2400

  ✓ pet created
  ✗ status is 200
    ↳  98% — ✓ 1182 / ✗ 18

  HTTP
  http_req_duration.......................................................: avg=6.62ms min=1.75ms med=5.49ms max=56.96ms p(90)=10.15ms p(95)=13.17ms
    { expected_response:true }............................................: avg=6.63ms min=1.75ms med=5.5ms  max=56.96ms p(90)=10.14ms p(95)=13.17ms
  http_req_failed.........................................................: 0.75%  18 out of 2400
  http_reqs...............................................................: 2400   39.3882/s

  EXECUTION
  iteration_duration......................................................: avg=1.01s  min=1s     med=1.01s  max=1.09s   p(90)=1.01s   p(95)=1.02s
  iterations..............................................................: 1200   19.6941/s
  vus.....................................................................: 20     min=20         max=20
  vus_max.................................................................: 20     min=20         max=20

  NETWORK
  data_received...........................................................: 189 MB 3.1 MB/s
  data_sent...............................................................: 503 kB 8.3 kB/s
```

Performance Testing Report

1. Introduction

The purpose of this load test is to evaluate the performance of the API under moderate load conditions. The goal is to identify any potential performance issues that may arise when the API is subjected to a reasonable number of concurrent requests.

2. Test Environment

Tool Used: k6

Number of Virtual Users (VUs): 20

Test Duration: 1 minute

Main Script: LoadTest.js

3. Test Scenarios
The load test was conducted to evaluate the performance of the following scenarios:

Pet Creation: Continuous POST requests were made to create new pets.

Fetching Pets: GET requests were made to retrieve the list of available pets.

4. Results
4.1 Summary of Checks
Total Checks Performed: 2400

Successful Checks: 2382 (99.25%)

Failed Checks: 18 (0.75%)

4.2 Specific Checks
Pet Created:

Successes: 1182

Failures: 18

Status Code 200:

Successes: 1164

Failures: 18

4.3 HTTP Request Metrics
Average Request Duration: 6.62 ms

Minimum Request Duration: 1.75 ms

Median Request Duration: 5.49 ms

Maximum Request Duration: 56.96 ms

90th Percentile: 10.15 ms

95th Percentile: 13.17 ms

HTTP Requests Failed: 0.75% (18 out of 2400)

4.4 Execution Metrics

Average Iteration Duration: 1.01 s

Total Iterations Completed: 1200

Virtual Users (VUs):

Minimum: 20

Maximum: 20

4.5 Network Metrics

Data Received: 189 MB (3.1 MB/s)

Data Sent: 503 kB (8.3 kB/s)

5. Analysis

Success Rate: The success rate of 99.25% is high, but the 0.75% of failures in status code checks indicates that some requests are not being processed correctly. This may signal issues on the server, such as timeouts or connection problems.

Request Duration: The average request duration of 6.62 ms is well within acceptable limits, with a maximum duration of 56.96 ms. While the average is low, the maximum duration suggests that there may be occasional spikes in response time that should be monitored.

Network Performance: The data transfer rates indicate that the API is capable of handling significant amounts of data efficiently.

6. Recommendations

Investigate Errors: Review server logs to identify the cause of the errors in requests that did not return a status code of 200.

Performance Optimization: Investigate the occasional spikes in request duration to identify potential bottlenecks in the API or backend services.

Load Testing with Increased VUs: Consider conducting additional tests with a higher number of virtual users to evaluate how the API performs under heavier loads.

Continuous Monitoring: Implement continuous monitoring to observe system performance in production and detect issues before they affect users.

Documentation of Results: Keep a record of these results for future reference and to compare with subsequent tests to identify trends and performance improvements.