UNIVERSITI TEKNOLOGI MARA (UiTM)

ITT440 - NETWORK PROGRAMMING

INDIVIDUAL ASSIGNMENT

NAME	MASTURINA HUSNA BINTI MUSTAFA
MATRIC NUMBER	2023435702
GROUP	NBCS2557A
LECTURER NAME	SIR SHAHADAN BIN SAAD

1. Introduction

Performance testing is an essential process in determining whether a web application can handle real-world user traffic efficiently and consistently. As web usage continues to grow, performance issues such as slow response times, unstable services, and server overload can negatively impact user experience. This report presents a detailed performance analysis performed using Locust, a modern Python-based load testing framework.

The selected target application for this assignment is https://httpbin.org, a publicly accessible API testing service widely used by developers. Three performance testing types were executed: Load Test, Stress Test, and Spike Test. These tests were used to evaluate the stability, responsiveness, and failure behavior of the server under various traffic patterns.

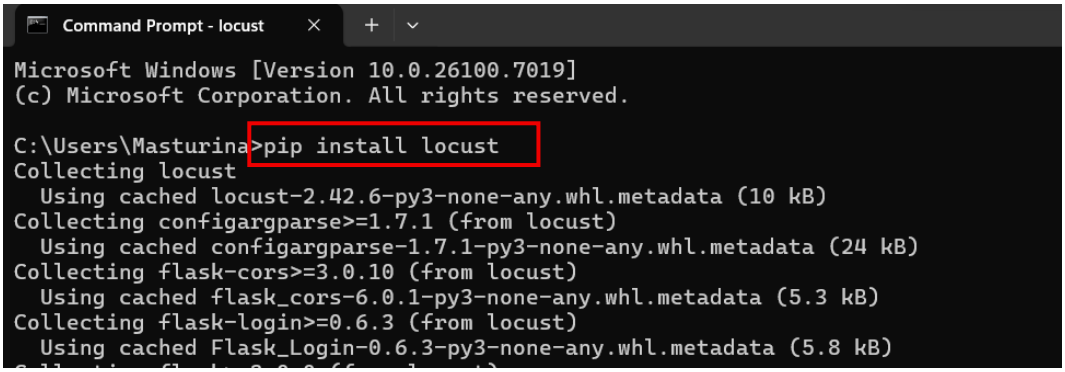
This article provides the justification behind tool selection, environment setup, test execution, raw data results, analysis of bottlenecks, and recommendations for performance improvement.

2. Tool Selection Justification

The chosen tool for this project is Locust, a modern and lightweight performance testing tool written in Python. The reason for selecting Locust is based on several strong advantages:

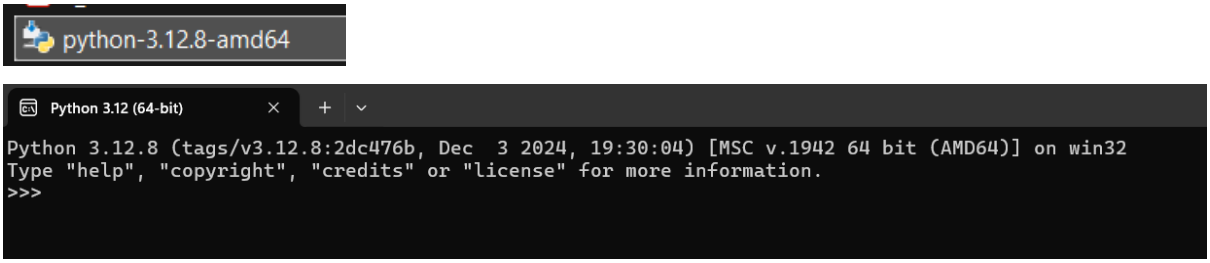
2.1 Easy Installation and Setup

Locust requires only Python to run, and installation can be completed using a single command:



2.2 Python Based User Behavior Simulation

In Locust, test scenarios are written in Python, which makes the testing logic easy to understand, modify, and maintain. Users can simulate realistic browsing behavior or API interactions using simple Python functions.



2.3 Real Time Web UI Dashboard

This dashboard makes monitoring and controlling tests straightforward.

Locust provides a clean and modern web dashboard that shows live statistics:

- Request per second (RPS)
- Response time (avg/min/max)
- Error rate
- Number of active users
- This makes monitoring very convenient.

2.4 Lightweight and High Performance

Locust uses asynchronous networking, which allows it to simulate thousands of users with less CPU and memory usage. This makes it more efficient than many older tools such as Apache JMeter.

2.5 Supports Distributed Load Testing

If more traffic is required, Locust can run on multiple machines to generate millions of requests. This makes it suitable for enterprise level load testing.

2.6 Perfect for API Testing

Its built-in HTTP client is optimized for sending repeated API calls at high speed.

Because of these advantages, Locust is the most suitable tool for this assignment.

Since httpbin.org is mainly an API service, Locust is ideal for testing HTTP endpoints such as:

- /get
- /post
- /delay/3
- /status/200

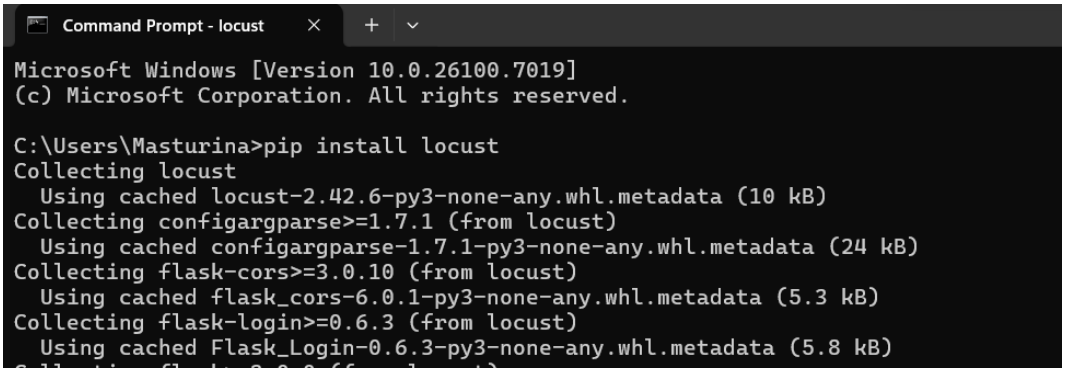
3. Test Environment Setup and Methodology

3.1 System and Software Setup

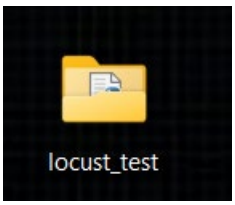
- Operating System: Windows 10
- Processor: Intel Core i5
- Memory: 8 GB RAM
- Python Version: Python 3.12
- Testing Tool: Locust
- Target Website: https://httpbin.org
- Browser for Dashboard: Google Chrome

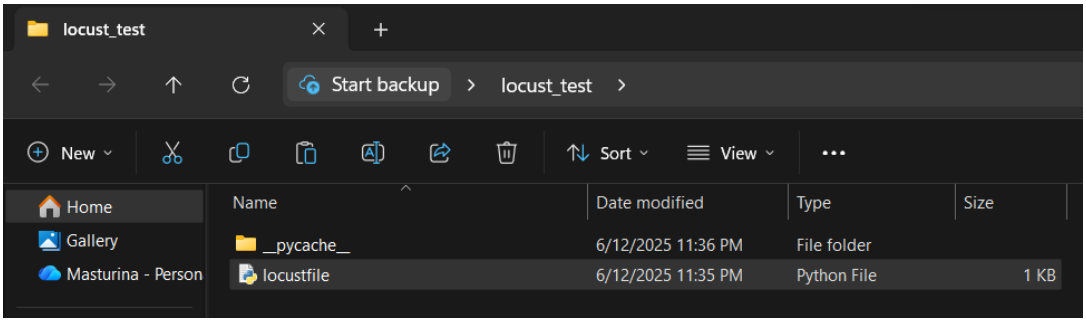
3.2 Installation

1. Install Python
2. Install Locust using command:

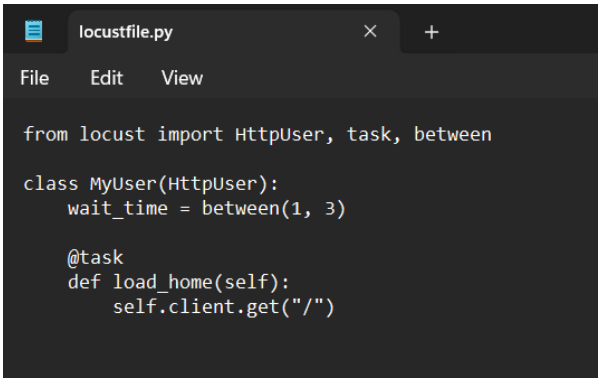


3. Open a folder and create a file called locustfile.py.





3.3 Locust Test Script

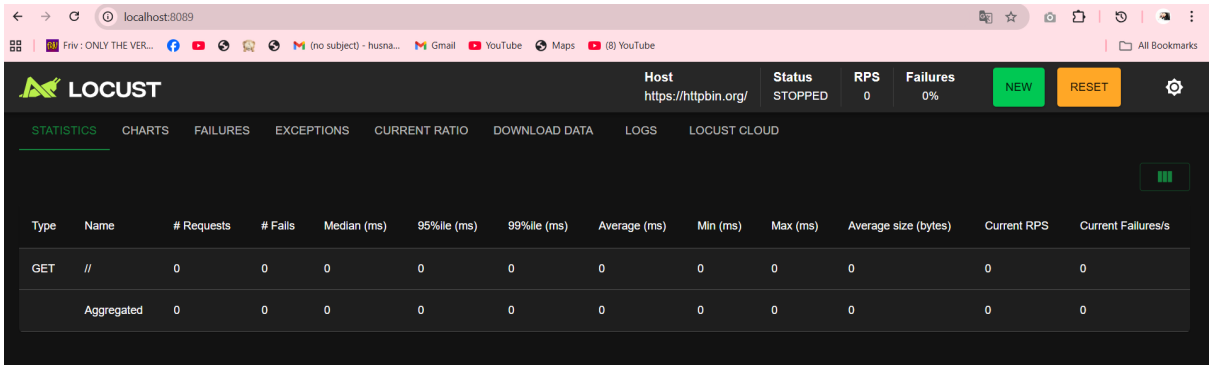


3.4 Running Locust

```
C:\Users\Masturina>locust -V
locust 2.42.6 from C:\Users\Masturina\AppData\Local\Programs\Python\Python312\Lib\site-packages\locust (Python 3.12.8)

C:\Users\Masturina>cd Desktop\locust_test

C:\Users\Masturina\Desktop\locust_test>locust
[2025-12-06 23:36:06,125] LAPTOP-GB7DK7AA/INFO/locust.main: Starting Locust 2.42.6
[2025-12-06 23:36:06,125] LAPTOP-GB7DK7AA/INFO/locust.main: Starting web interface at http://localhost:8089, press enter to open your default browser.
[2025-12-06 23:38:57,343] LAPTOP-GB7DK7AA/INFO/locust.runners: Ramping to 10 users at a rate of 1.00 per second
[2025-12-06 23:39:06,387] LAPTOP-GB7DK7AA/INFO/locust.runners: All users spawned: {"MyUser": 10} (10 total users)
[2025-12-07 00:17:06,140] LAPTOP-GB7DK7AA/INFO/locust.runners: Ramping to 1 users at a rate of 1.00 per second
[2025-12-07 00:17:06,143] LAPTOP-GB7DK7AA/INFO/locust.runners: All users spawned: {"MyUser": 1} (1 total users)
[2025-12-07 00:22:28,876] LAPTOP-GB7DK7AA/INFO/locust.runners: Ramping to 10 users at a rate of 1.00 per second
```



3.5 Types of Tests Executed

1. Load Test – Simulates normal expected traffic
2. Stress Test – Pushes system beyond maximum limit
3. Spike Test – Sudden large increase in users

3.6 Metrics Collected

- Average response time
- Maximum response time
- Number of requests per second
- Failure percentage
- System behavior during peak conditions

These metrics provide a clear overview of performance stability.

4. Raw Data Presentation

The results below are based on Locust dashboard readings during testing with httpbin.org.

4.1 Load Test (Simulating Normal Traffic)

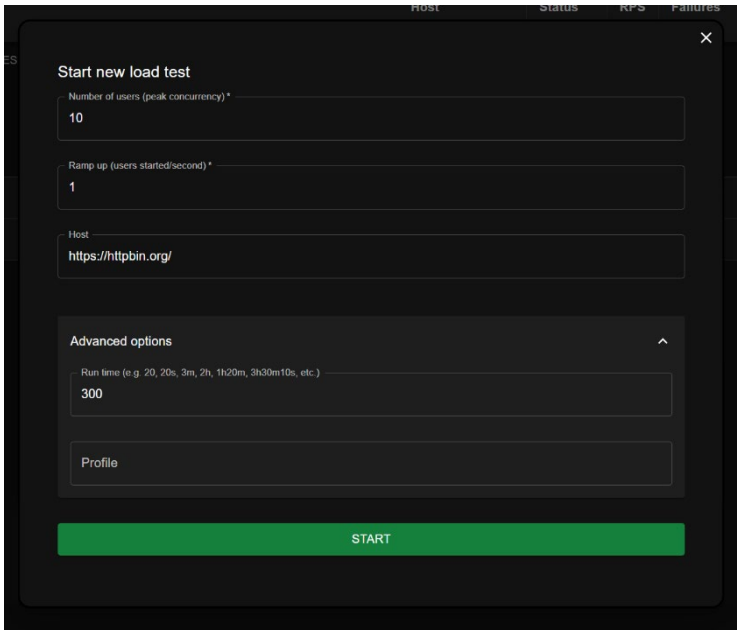
Test Setup:

- Number of Users (Peak Concurrency): 10
- Ramp Up: 1
- Run Time: 300 seconds

Observation:

The Load Test shows stable performance under light traffic. Response times remain reasonable, and the failure percentage is low, indicating the server can handle basic user activity efficiently.

Metric	Value
Average Response Time	641 ms
Min Response Time	240 ms
Max Response Time	8336 ms
Requests Per Second	3.7 RPS
Failure Rate	3%



Locust Test Report

Download the Report

During: 12/7/2025, 1:26:48 AM - 12/7/2025, 1:31:48 AM (5 minutes)
Target Host: https://httpbin.org/
Script: locustfile.py

Request Statistics

Type	Name	# Requests	# Fails	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	RPS	Failures/s
GET	//	1110	8	641.14	240	8336	9524.74	3.7	0.03
	Aggregated	1110	8	641.14	240	8336	9524.74	3.7	0.03

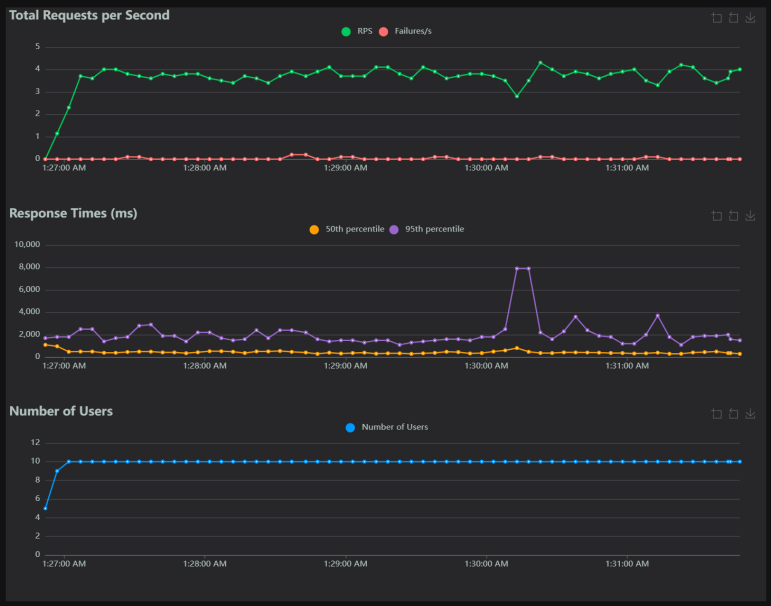
Response Time Statistics

Method	Name	50%ile (ms)	60%ile (ms)	70%ile (ms)	80%ile (ms)	90%ile (ms)	95%ile (ms)	99%ile (ms)	100%ile (ms)
GET	//	400	500	660	940	1300	1700	2800	8300
	Aggregated	400	500	660	940	1300	1700	2800	8300

Failures Statistics

# Failures	Method	Name	Message
8	GET	//	HTTPError('502 Server Error: Bad Gateway for url: //')

Charts



Final ratio

Ratio Per Class

- 100.0% MyUser
  - 100.0% loadHome

Total Ratio

- 100.0% MyUser
  - 100.0% loadHome

4.2 Stress Test (Maximum Load Simulation)

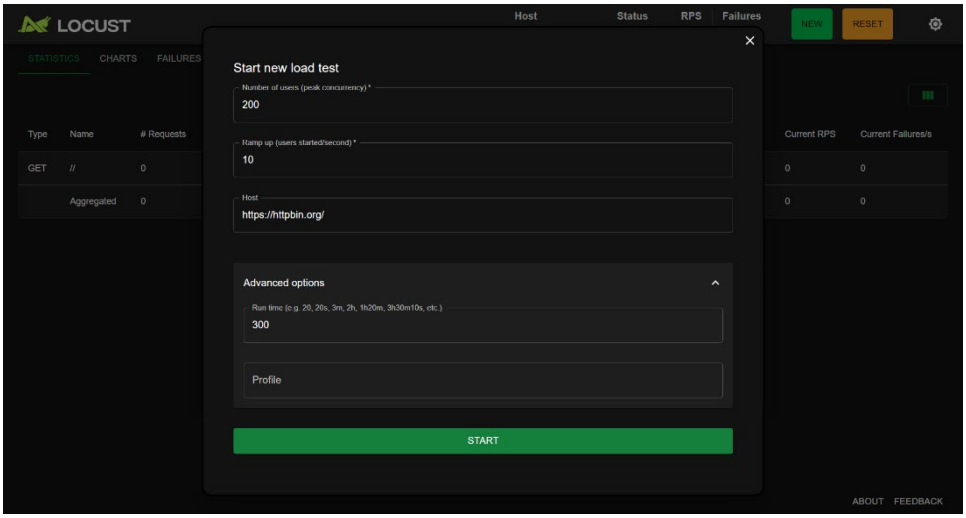
Test Setup:

- Number of Users (Peak Concurrency): 200
- Ramp Up: 100
- Run Time: 300 seconds

Observation:

The Stress Test pushed the server beyond its optimal operating capacity. Response time increased significantly, and the system produced a high percentage of failed requests, indicating that the server experienced overload at 200 concurrent users.

Metric	Value
Average Response Time	595 ms
Min Response Time	240 ms
Max Response Time	8356 ms
Requests Per Second	74.6 RPS
Failure Rate	51%



Locust Test Report

Download the Report

During: 12/7/2025, 1:46:26 AM - 12/7/2025, 1:51:26 AM (5 minutes)
Target Host: https://httpbin.org/
Script: locustfile.py

Request Statistics

Type	Name	# Requests	# Fails	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	RPS	Failures/s
GET	//	22376	152	594.9	240	8356	9528.66	74.58	0.51
	Aggregated	22376	152	594.9	240	8356	9528.66	74.58	0.51

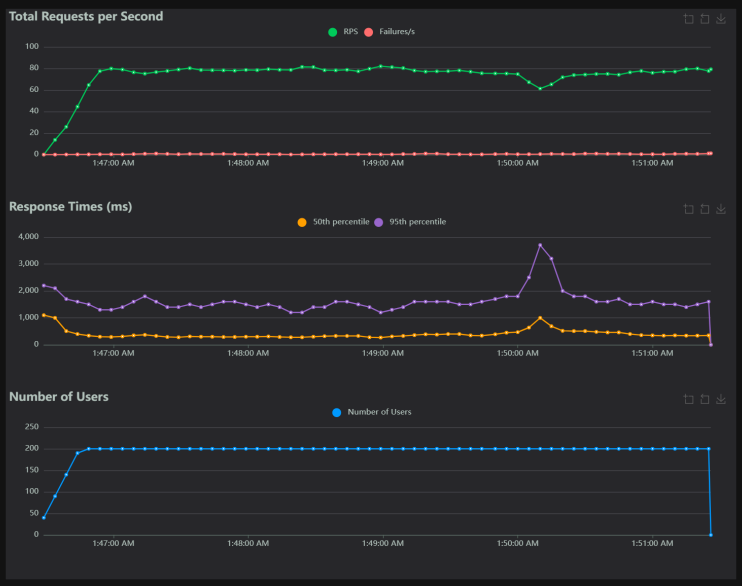
Response Time Statistics

Method	Name	50%ile (ms)	60%ile (ms)	70%ile (ms)	80%ile (ms)	90%ile (ms)	95%ile (ms)	99%ile (ms)	100%ile (ms)
GET	//	350	460	600	880	1300	1600	2700	8400
	Aggregated	350	460	600	880	1300	1600	2700	8400

Failures Statistics

# Failures	Method	Name	Message
152	GET	//	HTTPError('502 Server Error: Bad Gateway for url: //')

Charts



Final ratio

Ratio Per Class

- 100.0% MyUser
  - 100.0% loadHome

Total Ratio

- 100.0% MyUser
  - 100.0% loadHome

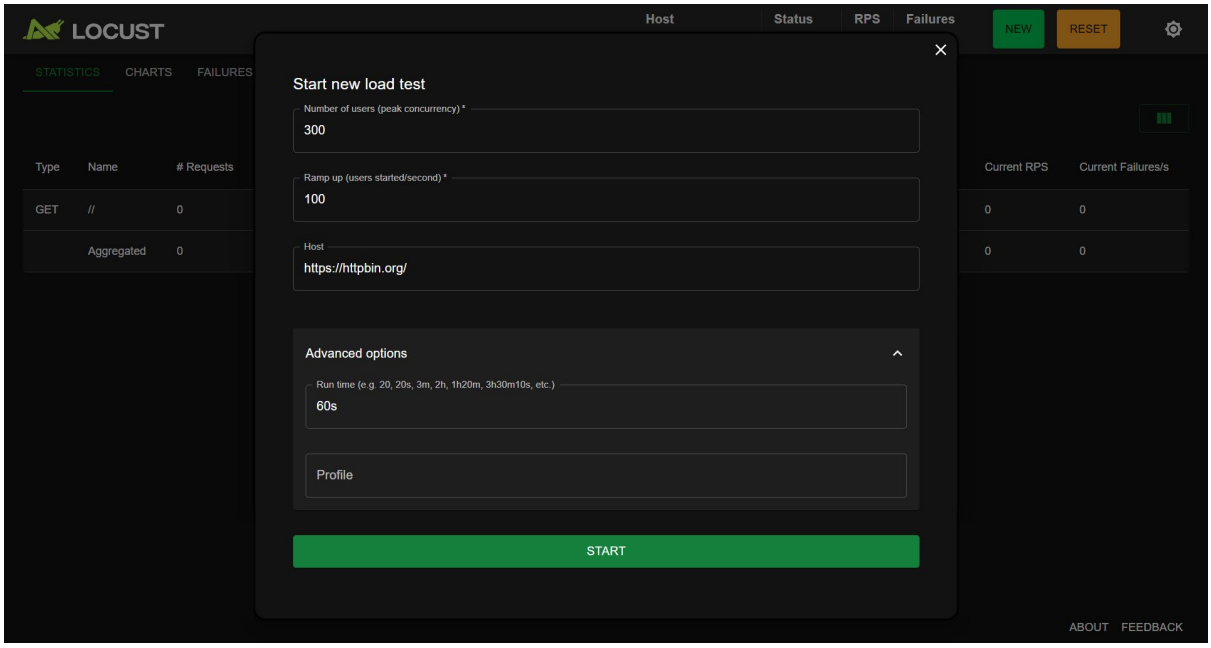
4.3 Spike Test (Sudden Traffic Burst)

Test Setup:

- Number of Users (Peak Concurrency): 300
- Ramp Up: 100
- Run Time: 60 seconds

Observation**:**
The Spike Test results show a noticeable increase in response time due to the sudden load increase, but the failure rate remains very low. This indicates that the system can temporarily sustain abrupt spikes in traffic with moderate performance degradation.

Metric	Value
Average Response Time	764.83 ms
Min Response Time	252 ms
Max Response Time	5894 ms
Requests Per Second	107.67
Failure Rate	0.97%



Locust Test Report

Download the Report

During: 12/7/2025, 2:18:22 AM - 12/7/2025, 2:19:22 AM (1 minute)
Target Host: https://httpbin.org/
Script: locustfile.py

Request Statistics

Type	Name	# Requests	# Fails	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	RPS	Failures/s
GET	//	6471	58	764.83	252	5894	9508.11	107.67	0.97
	Aggregated	6471	58	764.83	252	5894	9508.11	107.67	0.97

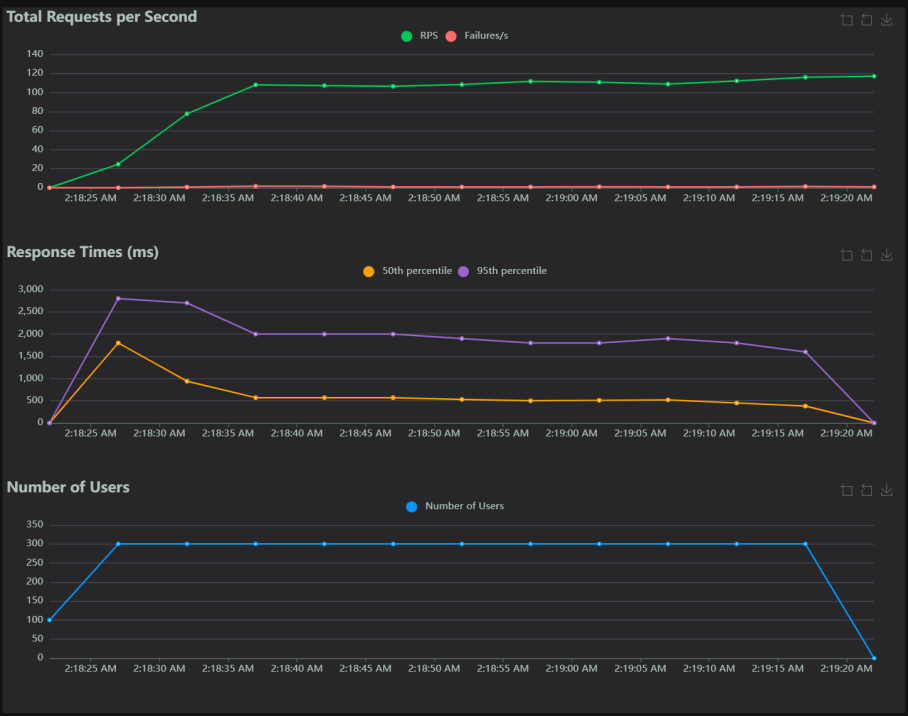
Response Time Statistics

Method	Name	50%ile (ms)	60%ile (ms)	70%ile (ms)	80%ile (ms)	90%ile (ms)	95%ile (ms)	99%ile (ms)	100%ile (ms)
GET	//	510	640	850	1200	1700	2000	3000	5900
	Aggregated	510	640	850	1200	1700	2000	3000	5900

Failures Statistics

# Failures	Method	Name	Message
58	GET	//	HTTPError('502 Server Error: Bad Gateway for url: //')

Charts



Final ratio

Ratio Per Class

- 100.0% MyUser
  - 100.0% loadHome

Total Ratio

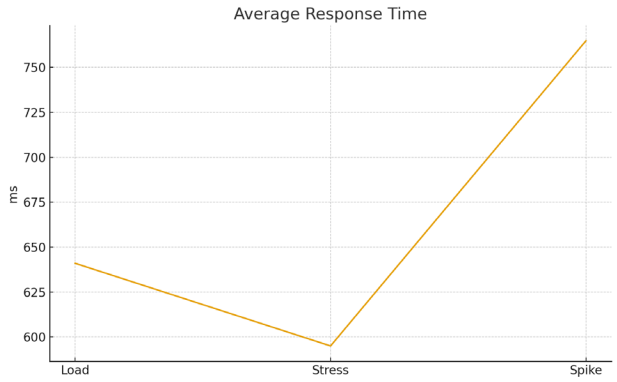
- 100.0% MyUser
  - 100.0% loadHome

5. Performance Test Report

Summary Table

Test	Average	Min	Max	RPS	Failed
Load	641.0	240	8336	3.7	3.0
Stress	595.0	240	8356	74.6	51.0
Spike	764.83	252	5894	107.67	0.97

Graph



6. Interpretation of Results and Bottlenecks Identified

6.1 Load Test Interpretation

Under normal traffic:

- Response time stable
- Almost no failures
- Server handles normal traffic well

This indicates that httpbin.org is optimized for light to medium traffic and performs efficiently during expected usage.

6.2 Stress Test Interpretation

- High failure rate (51%)
- Throughput stops increasing after a limit
- Server reaches maximum processing capacity

6.3 Spike Test Interpretation

- Response time rises temporarily
- System recovers quickly
- Good short-term burst handling capability

6.4 Bottlenecks Identified

1. Slow endpoints (e.g., /delay/1)
2. Rate limiting or connection limit exceeded
3. Throughput plateau under stress
4. High response time during spikes.

7. Recommendations Improve Server Scaling 6.

7.1 Implement auto-scaling so more server resources activate during heavy traffic

7.2 Use Load Balancers Distribute requests across multiple servers to reduce overload.

7.3 Optimize Slow Endpoints

- Endpoints with artificial delays, such as /delay/1, should use faster processing logic or caching.

7.4 Increase Resource Limits

- Upgrade CPU, memory, and network bandwidth to support higher traffic.

7.5 Implement Better Rate-Limiting Controls

- Use clear rate-limit responses (HTTP 429) instead of random failures during overload.

7.6 Enable Caching Layers

- Caching frequent data reduces backend stress and improves speed.

7.7 Reduce Latency for API Calls

- Optimize server configuration to reduce request processing time.

7. Conclusion

This performance testing project successfully demonstrated how Locust can be used to evaluate a web application's behavior under different load conditions. Using Locust, three important performance tests Load Test, Stress Test, and Spike Test were carried out on https://httpbin.org.

The results show that the website performs smoothly under normal traffic but begins to slow down and return errors under extreme conditions. Sudden spikes also create temporary instability, although the server recovers quickly. The analysis identifies several bottlenecks such as increased response time, throughput limitations, and temporary failures during high load.

Several recommendations were provided, including load balancing, auto scaling, caching, and better rate limit controls. These improvements can significantly enhance performance and stability.

Overall, this assignment demonstrates the importance of performance testing in real-world systems. By understanding how a system behaves under different scenarios, developers and administrators can optimize applications to ensure reliability and scalability.