

- **Title: Performance Tuning**

- **Aim:** To do the performance tuning for Assignment No.3 and 4.

- **Introduction:**

When developing an application, it's important to know how it performs under various workloads. Performance software testing helps determine how responsive and stable an app will be in different scenarios.

Although testing is a key part of app development, many developers underestimate its importance. Skipping the testing phase could mean overlooking issues related to security, functionality, accessibility, and performance. **Performance testing is done to verify server response time and throughput under different load conditions.**

**Locust:**

- Locust is an easy to use, scriptable and scalable performance testing tool.
- You define the behaviour of your users in regular Python code, instead of being stuck in a UI or restrictive domain specific language.
- This makes Locust infinitely expandable and very developer friendly.

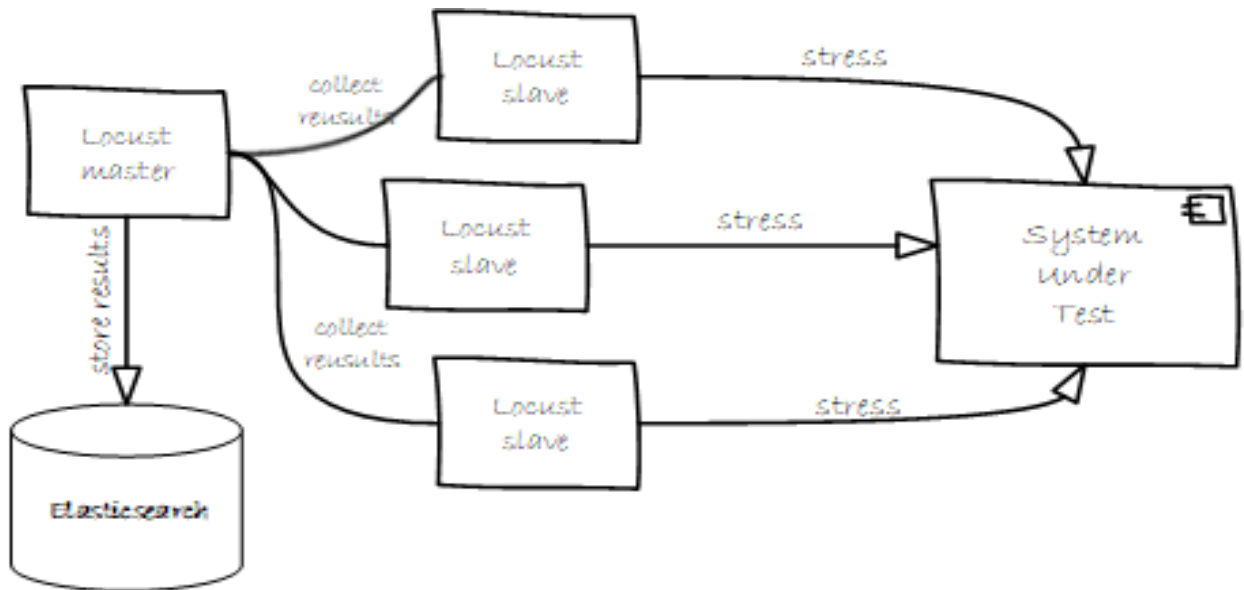
Locust has a user-friendly web interface that shows the progress of your test in real-time. You can even change the load while the test is running. It can also be run without the UI, making it easy to use for testing.

- **The Importance of Performance Testing:**

Performance testing is a crucial step in determining how your web app will perform under heavy loads. Locust is a valuable tool for performance testing, as it can discover the maximum number of users that your web app can handle.

- **Functional Block Diagram:**

The diagram below shows how Locust models users and simulates a heavy load:



- Creating a Performance Test in Locust

```
import time
from locust import HttpUser, task, between

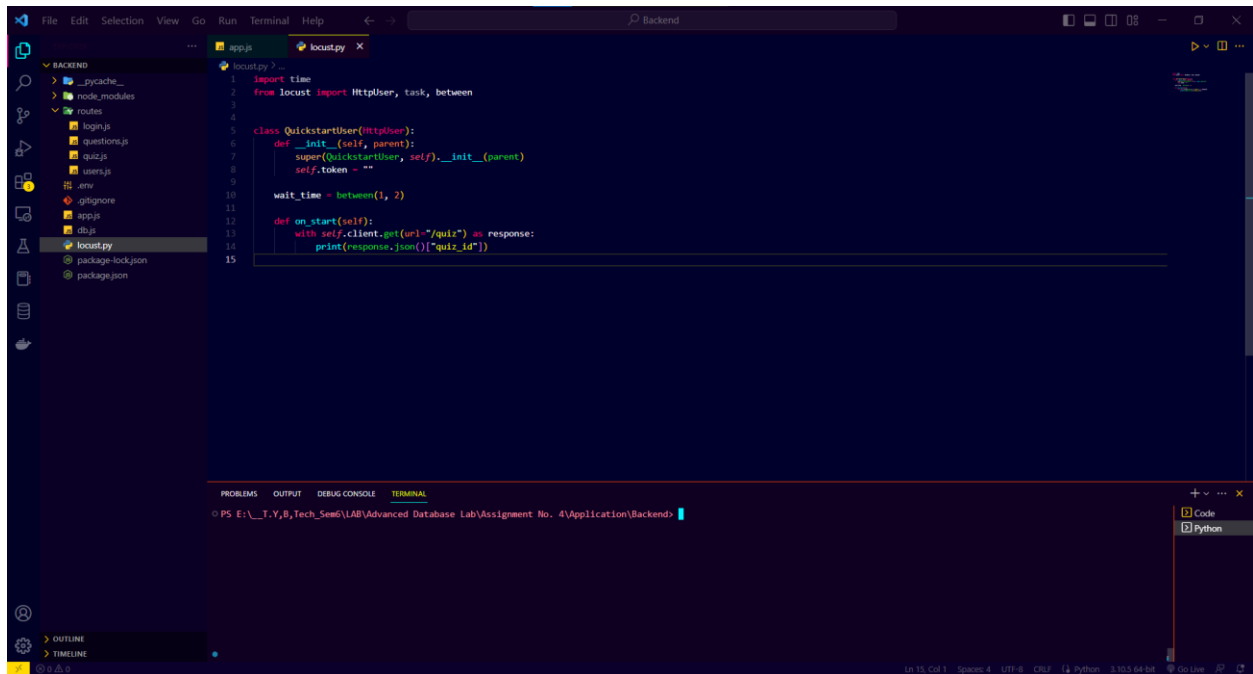
class QuickstartUser(HttpUser):
    def __init__(self, parent):
        super(QuickstartUser, self).__init__(parent)
        self.token = ""

    wait_time = between(1, 2)

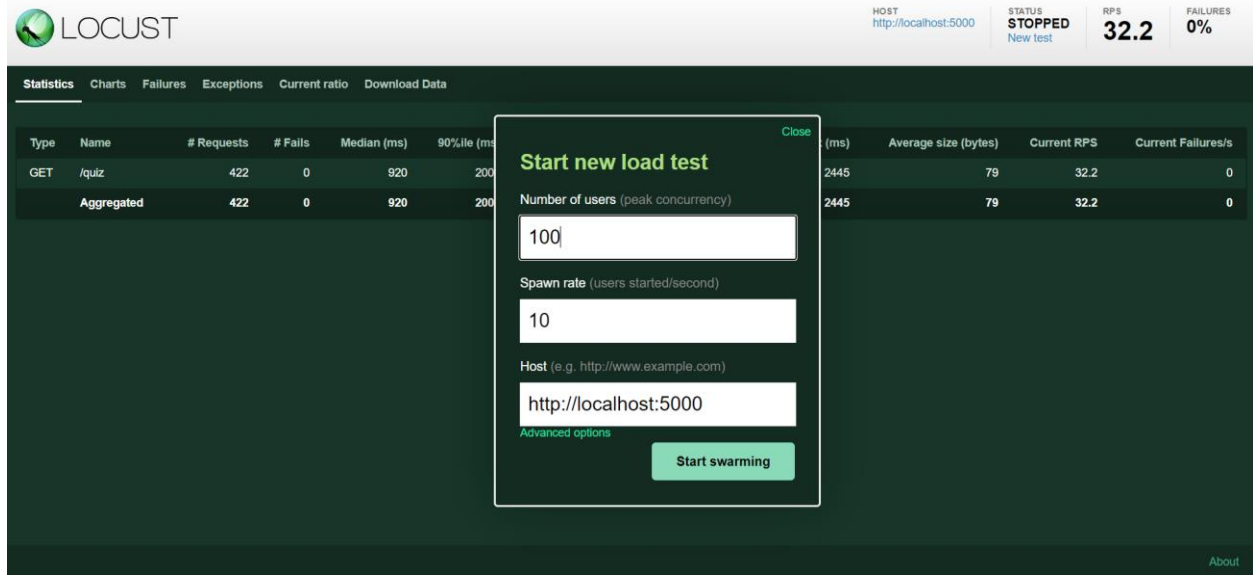
    def on_start(self):
        with self.client.get(url="/quiz") as response:
            print(response.json()["quiz_id"])

// locust -f locust.py
```

1. We created a performance test named “locust.py” for “basic load”



## 2. Making request for website's URL to make traffic



The Locust web interface shows a modal dialog for starting a new load test. The dialog contains the following fields and controls:

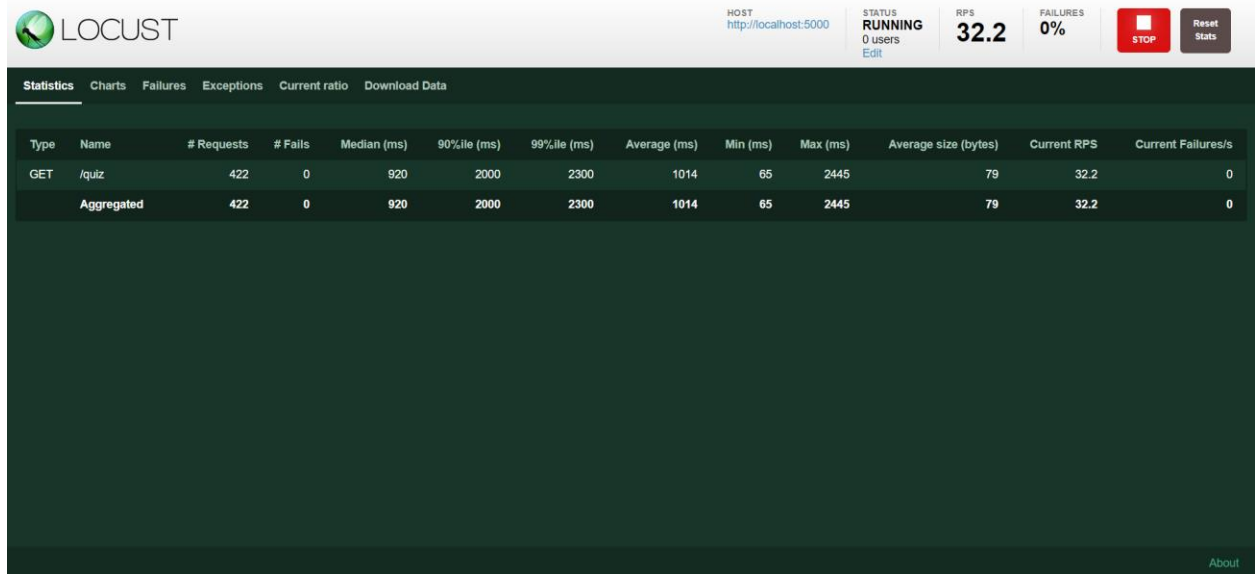
- Number of users (peak concurrency):** Input field with the value 100.
- Spawn rate (users started/second):** Input field with the value 10.
- Host (e.g. http://www.example.com):** Input field with the value http://localhost:5000.
- Advanced options:** A link to expand more settings.
- Start swarming:** A green button to initiate the test.

The background interface shows the following status and data:

- HOST:** http://localhost:5000
- STATUS:** STOPPED (with a link to 'New test')
- RPS:** 32.2
- FAILURES:** 0%
- Statistics table:**

Type	Name	# Requests	# Fails	Median (ms)	90%ile (ms)
GET	/quiz	422	0	920	2000
Aggregated		422	0	920	2000

## 3. Run the test and analyze the results:

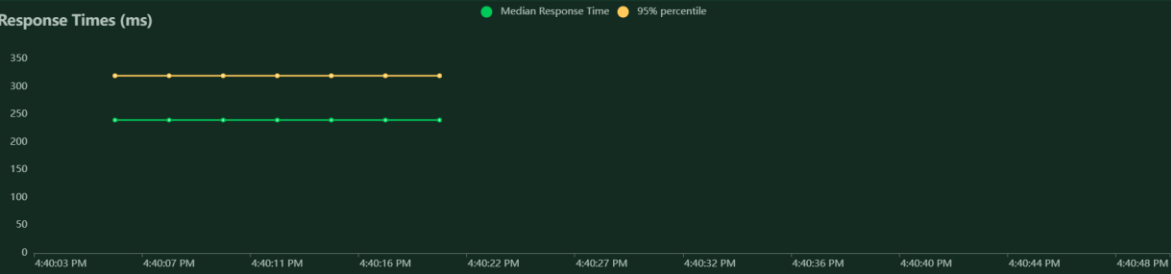


The Locust web interface shows the results of a running load test. The status and data are as follows:

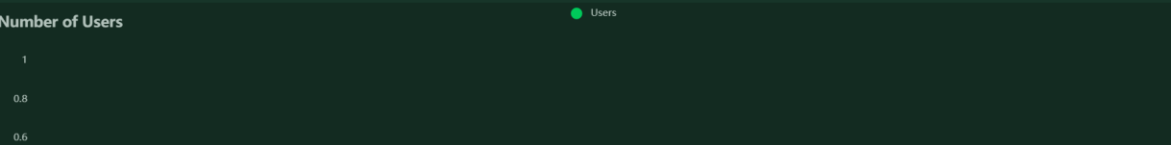
- HOST:** http://localhost:5000
- STATUS:** RUNNING (with a link to 'Edit') and 0 users
- RPS:** 32.2
- FAILURES:** 0%
- STOP button:** A red button to stop the test.
- Reset Stats button:** A grey button to reset the statistics.
- Statistics table:**

Type	Name	# Requests	# Fails	Median (ms)	90%ile (ms)	99%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s
GET	/quiz	422	0	920	2000	2300	1014	65	2445	79	32.2	0
Aggregated		422	0	920	2000	2300	1014	65	2445	79	32.2	0

### Response Times (ms)



### Number of Users



[Statistics](#)
[Charts](#)
[Failures](#)
[Exceptions](#)
[Current ratio](#)
[Download Data](#)

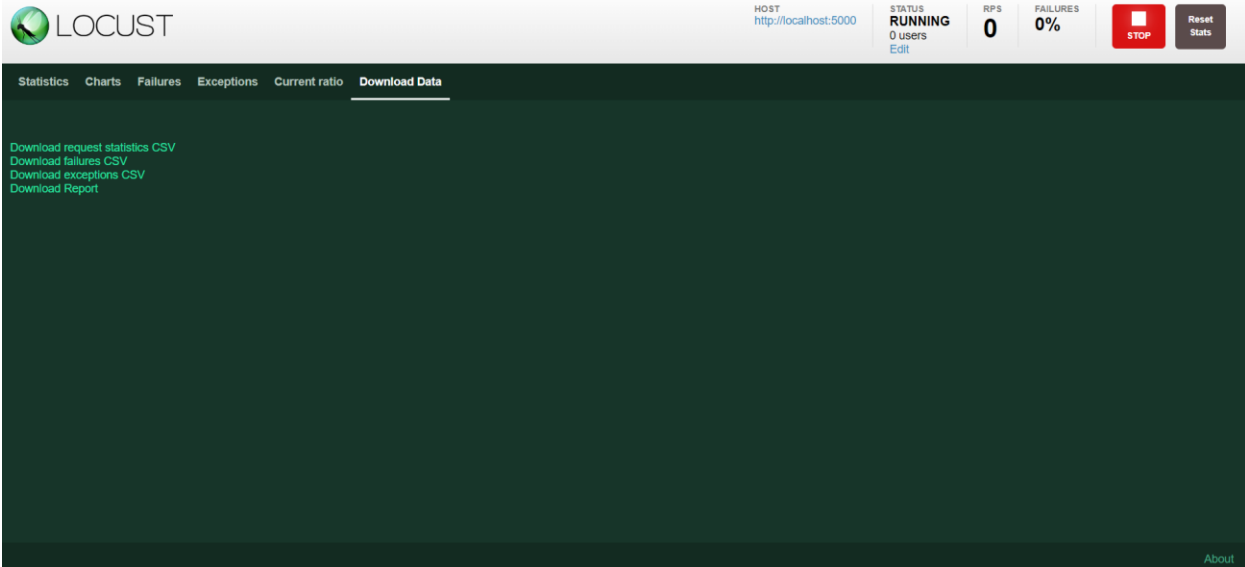
Script: locust.py

#### Ratio per User class

0.0% QuickstartUser  
100.0% secret\_page

#### Total ratio

0.0% QuickstartUser  
0.0% secret\_page



Carefully analyze the graph that you got in real time. Here are the two main factors to consider:

### **Throughput:**

Here we have average RPS is about 32.2% and failure is 0% for 100 users. The measuring capacity of the server, or how much it can handle. Ideally, this number should be infinite, if not very high.

It's important to note that throughput depends on other factors, such as internet speed, the Google server's current load, and CPU power. These factors continuously change, meaning you won't get the same results every time you run the test.

### **Deviation:**

The variation from the average. This number should be zero, or very low.

### **● Conclusion:**

After analyzing the graph, following are the conclusions:

- The throughput is 100% of input requests per minute.
- This means that the server handled all as 50 initially provided requests per minute.

- Sometimes, the throughput even got 1000 requests per minute. Based on these numbers, we can deduce that site has very good throughput.

- **References:**

<http://docs.locust.io/en/stable/what-is-locust.html>