Testing with Locust - Load Testing Tool Using python
-
Installation, Execution, Design and Reporting
A sample project

Rob Allan

Summary

1. Locust and Python Installation
2. System Under Test - Flask web server
3. System Under Test - Flask API server
4. Overview of Basic Web Load Tester
5. Overview of Basic API Load Tester
6. Running Locust from web GUI
7. Running Locust from Command Line
8. Code Location for all the project - at Rob’s github

Why - Supports Code based load testing using Python, lends itself to CICD integration and supports web and API based testing – as well as plugins for other protocols such as RPC and gRPC

Further info on Locust is available here … https://docs.locust.io/en/stable/

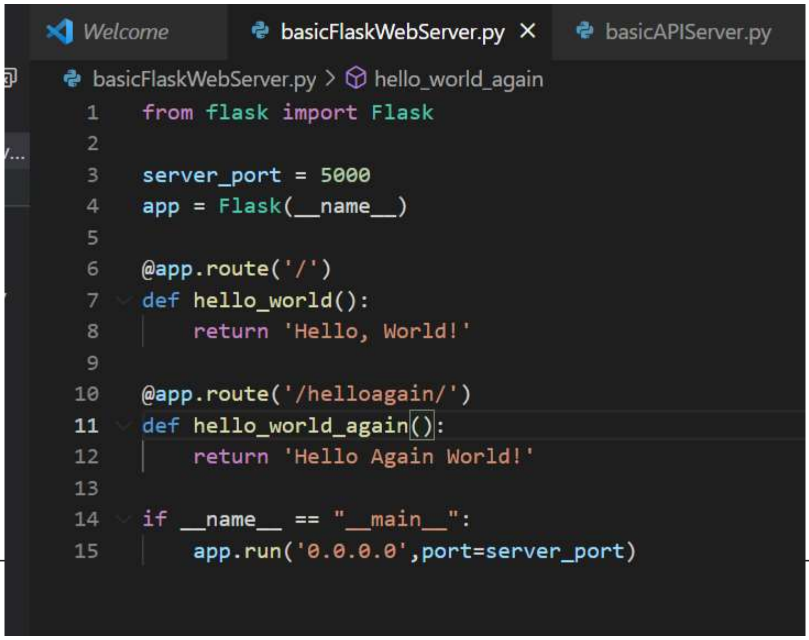
Locust and Python Installation

1. Insta;l Python – `python -v` to verify - tested against 3.9.6
2. This should include pip by default
3. Install locust - `pip install locust` – you may choose to sue an env to limit scope of packages
4. That’s it for install - easy…

System Under Test - Flask web server

1. For this testing a simple Flask web server is set up - to respond to specific URLs on a known port and limited set of routes – to run `python .\basicFlaskWebServer.py`

2. All code in github at end of this document.

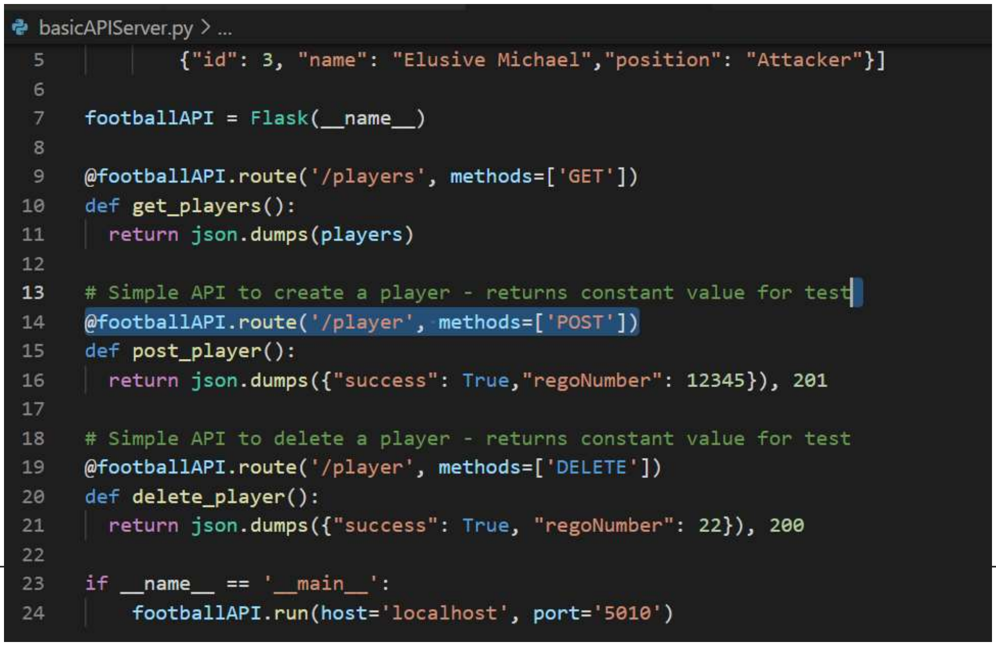


```python
from flask import Flask

server_port = 5000
app = Flask(__name__)

@app.route('/')
def hello_world():
    return 'Hello, World!'

@app.route('/helloagain/')
def hello_world_again():
    return 'Hello Again World!'

if __name__ == "__main__":
    app.run('0.0.0.0',port=server_port)
```

System Under Test - Flask API server

1. For this testing a simple Flask API server is set up - to respond to specific URLs on a known port and limited set of routes – also define JSON message bodies and response codes - to run `python .\basicAPIServer.py`

2. All code in github at end of this document.



basicAPIServer.py > ...

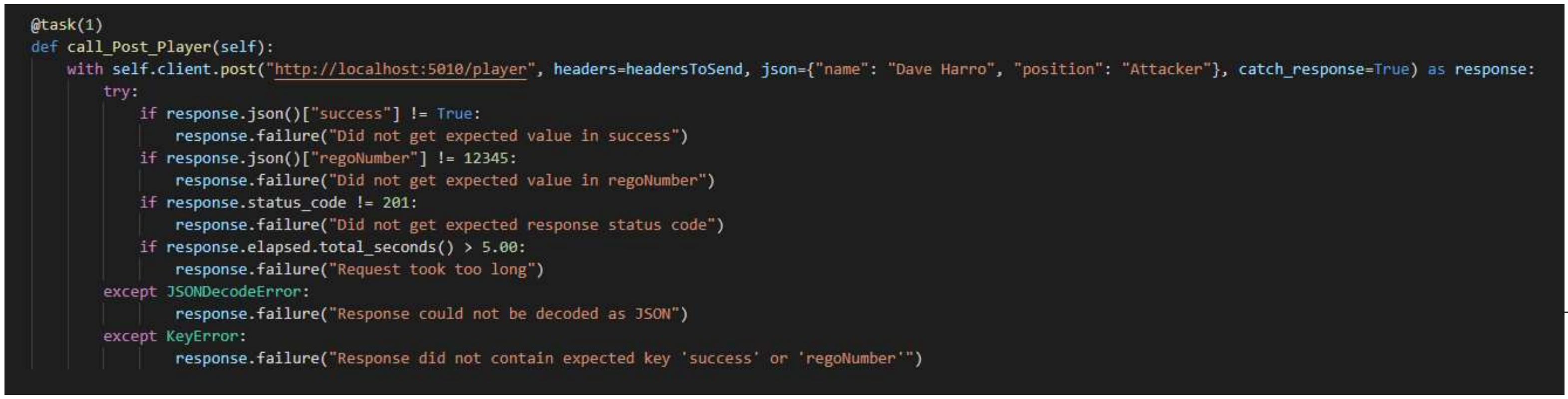
```python
        {"id": 3, "name": "Elusive Michael","position": "Attacker"}]

footballAPI = Flask(__name__)

@footballAPI.route('/players', methods=['GET'])
def get_players():
  return json.dumps(players)

# Simple API to create a player - returns constant value for test
@footballAPI.route('/player', methods=['POST'])
def post_player():
  return json.dumps({"success": True,"regoNumber": 12345}), 201

# Simple API to delete a player - returns constant value for test
@footballAPI.route('/player', methods=['DELETE'])
def delete_player():
  return json.dumps({"success": True, "regoNumber": 22}), 200

if __name__ == '__main__':
    footballAPI.run(host='localhost', port='5010')
```

Overview of Basic Web Load Tester

1. A simple file locustfile_web.py is provided.
2. Shows a simple file with several classes and with weighted task (more weighting more executions).
3. Shows the life cycle of tests with events before and after the complete cycle, or alternatively before and after each task
4. Calls to specific URLs, with validations for total time and text validations
5. Wait time can be used to control execution

Overview of Basic API Load Tester

1. A simple file locustfile_api.py is provided.
2. Shows a simple file with single classes and with equally weighted tasks (more weighting more executions).
3. Once again shows the life cycle of tests with events before and after the complete cycle, or alternatively before and after each task
4. Calls to specific URLs, with validations for total time, response code JSON text validations
5. Extract is shown below

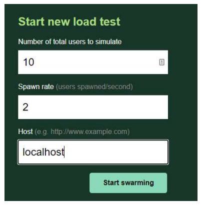


```python
@task(1)
def call_Post_Player(self):
    with self.client.post("http://localhost:5010/player", headers=headersToSend, json={"name": "Dave Harro", "position": "Attacker"}, catch_response=True) as response:
        try:
            if response.json()["success"] != True:
                response.failure("Did not get expected value in success")
            if response.json()["regoNumber"] != 12345:
                response.failure("Did not get expected value in regoNumber")
            if response.status_code != 201:
                response.failure("Did not get expected response status code")
            if response.elapsed.total_seconds() > 5.00:
                response.failure("Request took too long")
        except JSONDecodeError:
                response.failure("Response could not be decoded as JSON")
        except KeyError:
                response.failure("Response did not contain expected key 'success' or 'regoNumber'")
```

Running Locust from web GUI

The test cycles can be invoked to provoke a web GUI – typically on port 8089 - users can then control the number of threads and ramp up rate. Additionally the outputs can be examined- to invoke locust filename - here

- locust -f locustfile_api.py
- locust -f locustfile_web.py



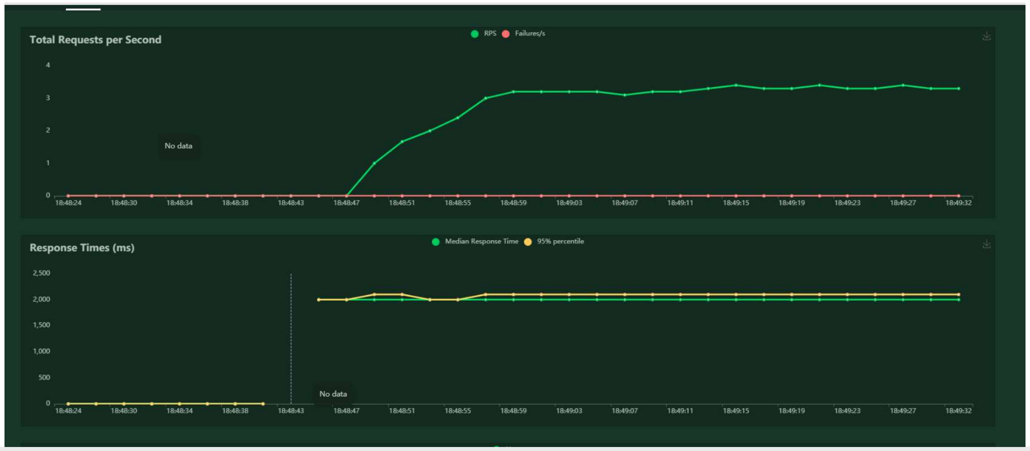
Start new load test

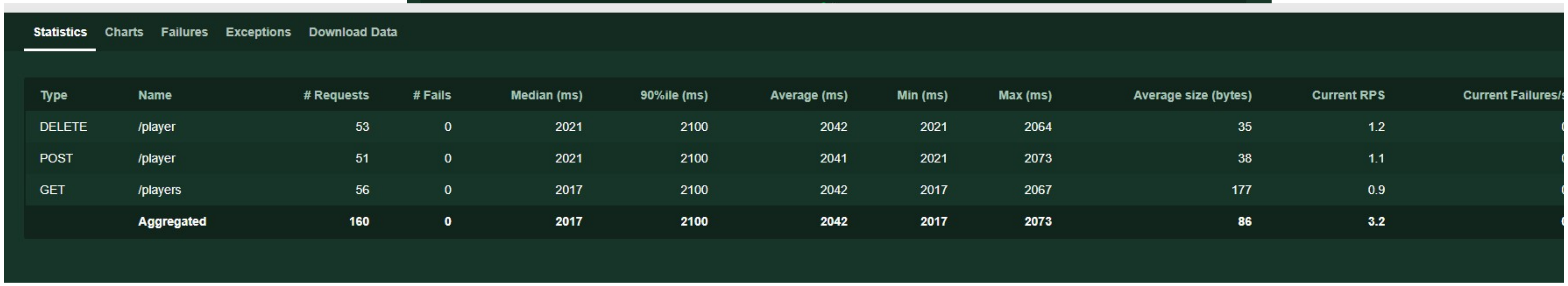
Number of total users to simulate: 10

Spawn rate (users spawned/second): 2

Host (e.g. http://www.example.com): localhost

Start swarming





Statistics | Charts | Failures | Exceptions | Download Data

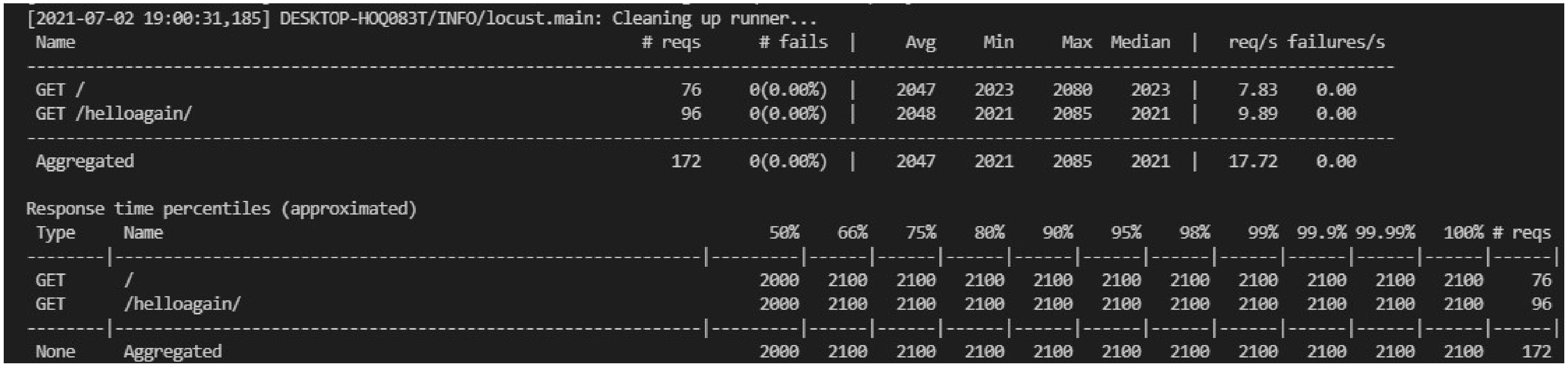
Type	Name	# Requests	# Fails	Median (ms)	90%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/
DELETE	/player	53	0	2021	2100	2042	2021	2064	35	1.2	0
POST	/player	51	0	2021	2100	2041	2021	2073	38	1.1	0
GET	/players	56	0	2017	2100	2042	2017	2067	177	0.9	0
	Aggregated	**160**	**0**	**2017**	**2100**	**2042**	**2017**	**2073**	**86**	**3.2**	**0**

Running Locust from Command Line

This will allows the execution with arguments. Headless typically for web cases. This example creates extra output example csv values, runs for 10 seconds, with up to 100 user with a ramp up rate of 10 per second.

```
locust -f locustfile_web.py --csv=example --headless -
-host wfng -u 100 -r 10 -t 10s
```

Further options through locust help



```
[2021-07-02 19:00:31,185] DESKTOP-HOQ083T/INFO/locust.main: Cleaning up runner...
 Name                                                          # reqs      # fails  |     Avg     Min     Max  Median  |   req/s failures/s
--------------------------------------------------------------------------------------------------------------------------------------------
 GET /                                                             76     0(0.00%)  |    2047    2023    2080    2023  |    7.83    0.00
 GET /helloagain/                                                  96     0(0.00%)  |    2048    2021    2085    2021  |    9.89    0.00
--------------------------------------------------------------------------------------------------------------------------------------------
 Aggregated                                                       172     0(0.00%)  |    2047    2021    2085    2021  |   17.72    0.00

Response time percentiles (approximated)
 Type     Name                                                              50%    66%    75%    80%    90%    95%    98%    99%  99.9% 99.99%   100% # reqs
--------|------------------------------------------------------------|---------|------|------|------|------|------|------|------|------|------|------|------|
 GET      /                                                                2000   2100   2100   2100   2100   2100   2100   2100   2100   2100   2100     76
 GET      /helloagain/                                                     2000   2100   2100   2100   2100   2100   2100   2100   2100   2100   2100     96
--------|------------------------------------------------------------|---------|------|------|------|------|------|------|------|------|------|------|------|
 None     Aggregated                                                       2000   2100   2100   2100   2100   2100   2100   2100   2100   2100   2100    172
```

Code Location for all the project - at Rob’s github