Performance Testing with Locust

A Python tool — Locust

JMeter in Java world.

Locust is totally coding based, there is no GUI option to write tests which itself is a big advantage with more flexibility, better control, faster test maintenance and can use version control system as well.

Locust is very light.

Locust can be installed using pip:

```
$ pip install locustio
```

Locust file is nothing but a plain Python file(e.g. mylocustfile.py) with some predefined classes and attributes in it.

```
from locust import HttpLocust, TaskSet, task, between

class TestCases(TaskSet):
    def on_start(self):
        self.payload = {"email": "john@example.com", "password":123}
        self.login()

    def on_stop(self):
        self.logout()
```

```
    def login(self):
        self.client.post("/login", self.payload)

    def logout(self):
        self.client.post("/logout", self.payload)

    @task(2)
    def visit_count(self):
        self.client.get("/visit")

    @task(1)
    def profile(self):
        self.client.get("/profile")

class LocustUsers(HttpLocust):
    task_set = TestCases
    wait_time = between(5, 9)
```

HttpLocust: it provides you the instance of *HttpSession through which we will be able to make Http calls to application server.*

TaskSet: Each performance test (which is called tasks in Locust) is collected under this TaskSet class as attribute and Locust will execute all of them during its test session. Understand this as a collection of performance test cases.

task: It is a decorator basically which converts a simple method into a performance test method or test case when used as a decorator. The moment you add @task decorator above any method, it becomes a test method(like a test case) for Locust.

between: This is optional. Consider a case where we have 10 tasks in script and want Locust to wait for a certain amount of time between executing those tasks one by one, then can use this Locust method. It takes two arguments e.g. *between(2, 5)* which means Locust will wait for a random time anywhere between 2 seconds to 5 seconds but that chosen wait will be uniform throughout the session.

TestCases: define a class that will inherit TaskSet class and can give this class any name

on_start(self): This method is provided by Locust which will be executed before any task sequence picks up. We see this also in the performance test report like any other task but don't need to call this explicitly as a task.

on_stop(self): Again, this is similar to the on_start method in nature but as the name indicates, it will be called and executed after a task sequence execution is done. e.g. want to logout at the end of each task set thread.

LocustUsers: Once we are done writing all tasks, need to define another class like this. we can consider this class acting like a real user which will be taking up all the tasks one by one and hitting application server. We must inherit the ***HttpLocust*** class provided by Locust itself.

task(1), task(2): Notice these numbers inside the task method? The meaning of these numbers is the task(2) will be executed twice for every execution of the task(1). We can keep this number as per our requirement. This gives the flexibility to execute tasks the way we want.

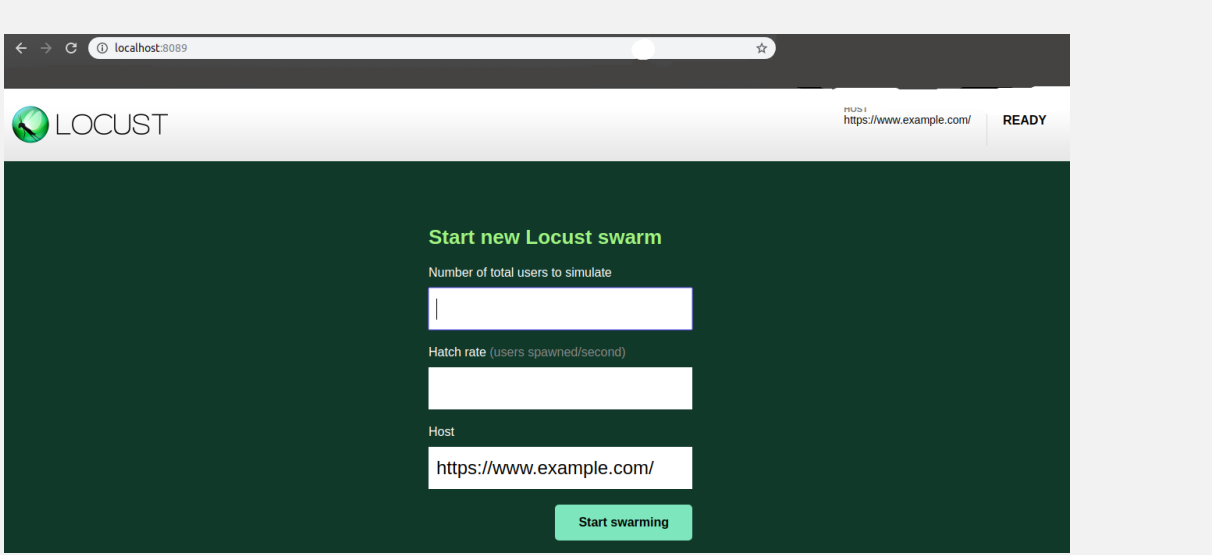
To run your Locust performance test script, run this command:

```
locust -f mylocustfile.py -- host= https://www.example.com/
```

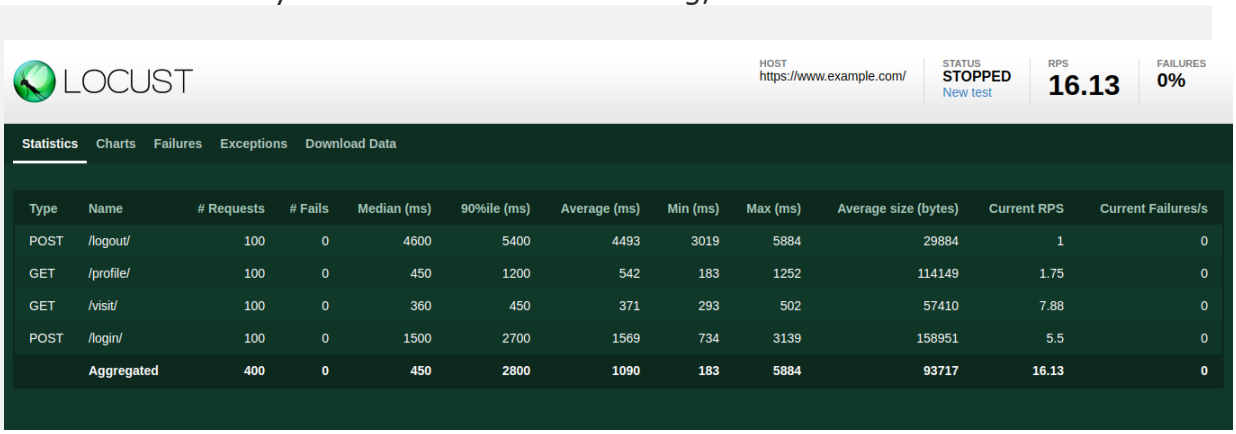
see something like this in your system console:

```
a ● locust -f dummy_test_creation_workflow.py --host='https://www.example.com/'
[2020-03-31 22:13:16,421]                    /INFO/locust.main: Starting web monitor at http://*:8089
[2020-03-31 22:13:16,422]                    /INFO/locust.main: Starting Locust 0.14.4
```

Now open any browser and navigate to http://localhost:8089. see a page like this:



In this UI, we can give the total number of users who will be using our application continuously(**Number of users to simulate**) and the number of users being activated per second(**Hatch Rate**). e.g If we give the number of users to simulate as 100 and Hatch rate as 10 then all 100 users will be activated within 10 seconds. Once you click on Starts Swarming, should see a UI like this:



Type	Name	# Requests	# Fails	Median (ms)	90%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s
POST	/logout/	100	0	4600	5400	4493	3019	5884	29884	1	0
GET	/profile/	100	0	450	1200	542	183	1252	114149	1.75	0
GET	/visit/	100	0	360	450	371	293	502	57410	7.88	0
POST	/login/	100	0	1500	2700	1569	734	3139	158951	5.5	0
	Aggregated	**400**	**0**	**450**	**2800**	**1090**	**183**	**5884**	**93717**	**16.13**	**0**

Once we are done with the testing, stop the session by clicking on the **Stop** button on the top right corner and then download the various reports available in the CSV format for further analysis.