

# Load Testing with Locust



**Dustin Ingram** (/author/dustin-ingram)

Director, Austin Office & Software Engineer

June 2, 2016

**Locust** (<https://locust.io/>) is an open source load-testing tool written in Python. It lets you write tests against your web application which mimic your user's behavior, and then run the tests at scale to help find bottlenecks or other performance issues. At PromptWorks, we use it regularly to make sure that the web applications we write are able to handle a high load and remain performant.

## Installation

Installation is done with Python's `pip` :

1

```
$ pip install locustio
```

## Configuration

One of the nicest features of Locust is that configuration is done via "Plain Old Python." You simply create a file named `locustfile.py` and all configuration for your load tester and its tests is done there.

Here's an example `locustfile.py` , which defines a simple user behavior which consists of a single "task" which gets a specific webpage:

Say hi

Send



```
1  from locust import HttpLocust, TaskSet, task
2
3  class UserBehavior(TaskSet):
4
5      @task
6      def get_something(self):
7          self.client.get("/something")
8
9  class WebsiteUser(HttpLocust):
10     task_set = UserBehavior
```

We can add a second task as follows:

```
1  class UserBehavior(TaskSet):
2
3      @task
4      def get_something(self):
5          self.client.get("/something")
6
7      @task
8      def get_something_else(self):
9          self.client.get("/something-else")
```

When the above `UserBehavior` is run, Locust will randomly choose between each of the tasks and run them. If you want to weight different tasks differently, so that one is run twice as much as the other, you can add weighting as follows:

```
1 class UserBehavior(TaskSet):
2
3     @task(2)
4     def get_something(self):
5         self.client.get("/something")
6
7     @task(1)
8     def get_something_else(self):
9         self.client.get("/something-else")
```

These task weights are ratios across all defined tasks, so here `get_something` would happen twice as often during the load test.

You can also write tasks which compose other tasks, to perform a sequential or serial set of tasks in a specific order. This lets you define a user flow through multiple requests. For example:<sup>1</sup>

```
1 class UserBehavior(TaskSet):
2
3     @task
4     def get_something(self):
5         self.client.get("/something")
6
7     @task
8     def get_something_else(self):
9         self.client.get("/something-else")
10
11    @task
12    def get_two_things(self):
13        self.get_something()
14        self.get_something_else()
```

A `TaskSet` class can optionally declare an `on_start` function, which is called when a simulated user starts executing that `TaskSet` class. This can be used to log in or apply credentials once before beginning the load test:

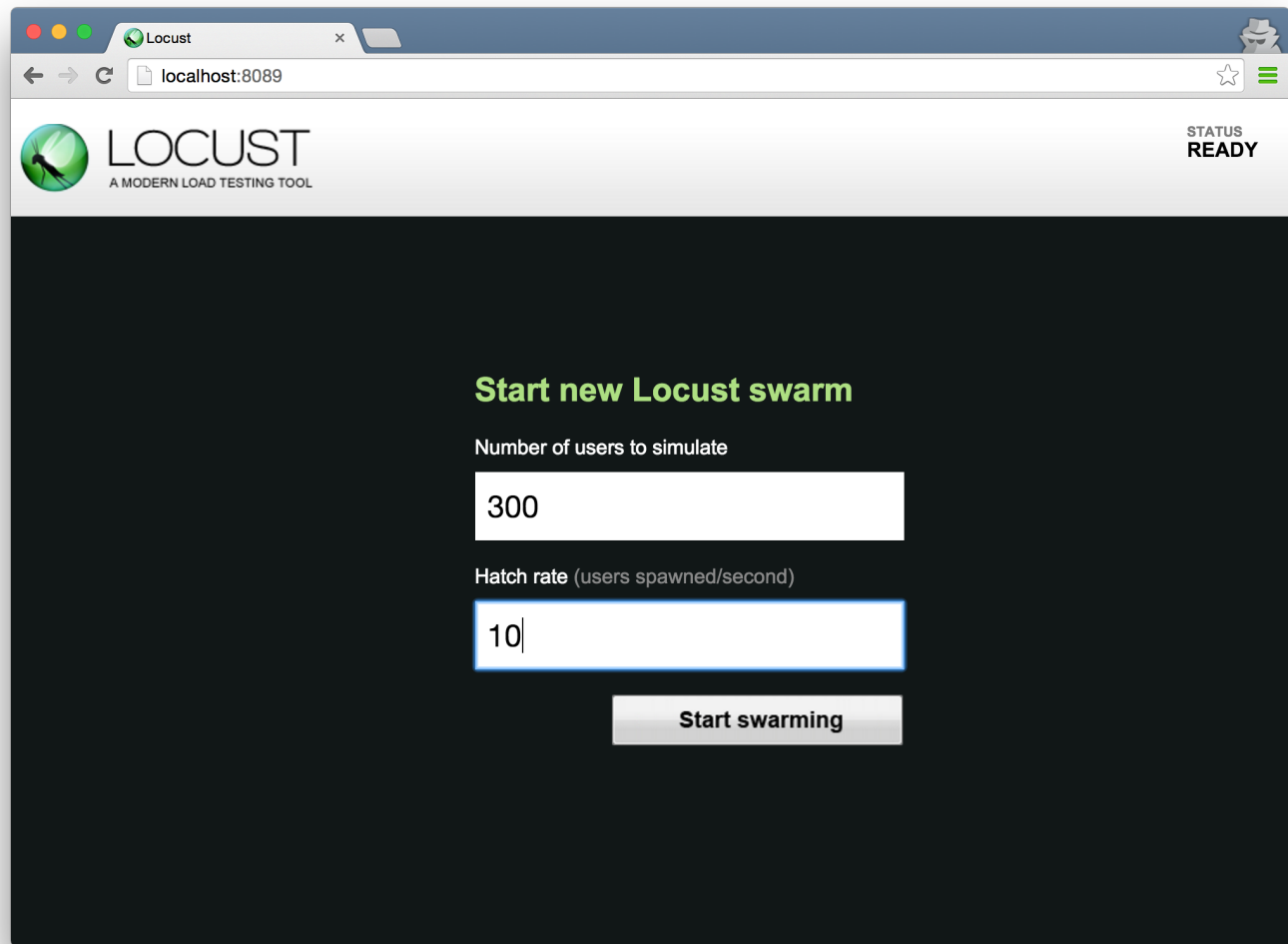
```
1 class UserBehavior(TaskSet):
2
3     def on_start(self):
4         self.client.post("/login", {
5             'username': 'foo', 'password': 'bar'
6         })
7
8     @task
9     def get_something(self):
10        self.client.get("/something")
```

## Running Locally

To run Locust, you run the `locust` command in the same directory as your `locustfile.py`:

```
1 $ locust --host=http://localhost:5000
```

Once the command is run, Locust starts up a local web server which you can visit in your browser:



Locust

localhost:8089

LOCUST  
A MODERN LOAD TESTING TOOL

STATUS  
READY

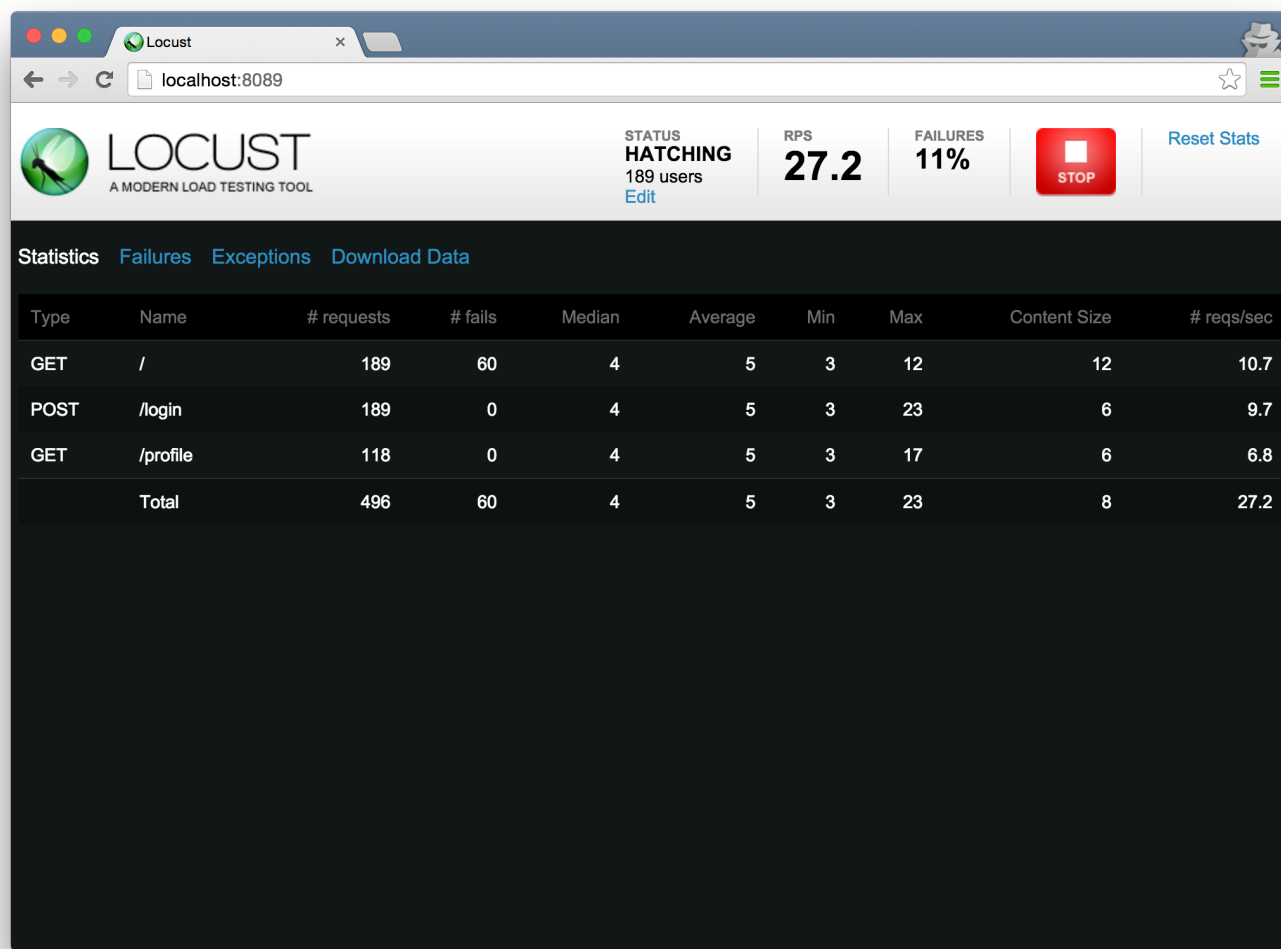
### Start new Locust swarm

Number of users to simulate

Hatch rate (users spawned/second)

Start swarming

After selecting the number of users and the spawn rate, you can begin the test, which will show you a live view of the running test:



## Running Distributed

Running locally is fine for basic testing and getting started with Locust, but most applications will not receive a significant load if you're just running it from your local machine. It's almost always necessary to run it in distributed mode. This is easy to do with a couple AWS nodes.

After installing Locust and moving your `locustfile.py` to all nodes, you can start the "master" node:

```
1 $ locust --host=http://localhost:5000 --master
```

Then start any "slave" nodes, giving them a reference to the master node:

```
1 $ locust --host=http://localhost:5000 --slave\  
2 --master-host=192.168.10.100
```

## Downsides

As nice as Locust is, there are a small number of downsides. First, statistics are pretty bad or nonexistent for the results of the test, and could be better (e.g., there are no graphs, and you can't correlate an increased failure rate with a higher load without running multiple tests). Second, it's sometimes hard to get details about error responses beyond the status. Finally, it's non-trivial to do non-HTTP or non-RESTful requests (although this is admittedly rare).

## Upsides

Overall, though Locust (<https://locust.io/>) is an incredibly useful load testing tool, especially for being an open-source project. If your codebase is Python, it's a shoe-in for the best tool you can be using, due to the opportunity to pull in data, models, or domain logic from your existing codebase, but even if you're not using Python, you can easily integrate it. Put your code to the test!

*Thanks to Brian Duggan and Patrick Smith for reading drafts of this post.*

---

1. Thanks to Andre for pointing out that this example was originally incorrect! ↩

---

**Test-driven:** Automated testing is a big part of [how our software engineers craft better software](#) ([/services/software-developers](#)) for companies of all kinds. [Learn More.](#) ([/services/software-developers](#))

---















