

# Quick start

## Example locustfile.py

Below is a quick little example of a simple **locustfile.py**:

```
from locust import HttpLocust, TaskSet, between

def login(l):
    l.client.post("/login", {"username": "ellen_key", "password": "education"})

def logout(l):
    l.client.post("/logout", {"username": "ellen_key", "password": "education"})

def index(l):
    l.client.get("/")

def profile(l):
    l.client.get("/profile")

class UserBehavior(TaskSet):
    tasks = {index: 2, profile: 1}

    def on_start(self):
        login(self)

    def on_stop(self):
        logout(self)

class WebsiteUser(HttpLocust):
    task_set = UserBehavior
    wait_time = between(5.0, 9.0)
```

Here we define a number of Locust tasks, which are normal Python callables that take one argument (a `Locust` class instance). These tasks are gathered under a `TaskSet` class in the `tasks` attribute. Then we have a `HttpLocust` class which represents a user, where we define how long a simulated user should wait between executing tasks, as well as what `TaskSet` class should define the user's "behaviour". `TaskSet` classes can be nested.

The `HttpLocust` class inherits from the `Locust` class, and it adds a `client` attribute which is an instance of `HttpSession` that can be used to make HTTP requests.

By default, we stop looking for proxy settings to improve performance. If you really want the test requests go through a HTTP proxy, you can inherit from the `HttpLocust` class and set the `trust_env` field to `True`. For further details, refer to the documentation of requests.

Another way we could declare tasks, which is usually more convenient, is to use the `@task` decorator. The following code is equivalent to the above:

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

class UserBehavior(TaskSet):
    def on_start(self):
        """ on_start is called when a Locust start before any task is scheduled """
        self.login()

    def on_stop(self):
        """ on_stop is called when the TaskSet is stopping """
        self.logout()

    def login(self):
        self.client.post("/login", {"username": "ellen_key", "password": "education"})

    def logout(self):
        self.client.post("/logout", {"username": "ellen_key", "password": "education"})

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

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

class WebsiteUser(HttpLocust):
    task_set = UserBehavior
    wait_time = between(5, 9)
```

The `Locust` class (as well as `HttpLocust` since it's a subclass) also allows one to specify the wait time between the execution of tasks ( `wait_time = between(5, 9)` ) as well as other user behaviours. With the `between` function the time is randomly chosen uniformly between the specified min and max values, but any user-defined time distributions can be used by setting `wait_time` to any arbitrary function. For example, for an exponentially distributed wait time with average of 1 second:

```
import random

class WebsiteUser(HttpLocust):
    task_set = UserBehaviour
    wait_time = lambda self: random.expovariate(1)*1000
```

## Start Locust

To run Locust with the above Locust file, if it was named `locustfile.py` and located in the current working directory, we could run:

```
$ locust
```

If the Locust file is located under a subdirectory and/or named different than *locustfile.py*, specify it using `-f`:

```
$ locust -f locust_files/my_locust_file.py
```

To run Locust distributed across multiple processes we would start a master process by specifying `--master`:

```
$ locust -f locust_files/my_locust_file.py --master
```

and then we would start an arbitrary number of slave processes:

```
$ locust -f locust_files/my_locust_file.py --slave
```

If we want to run Locust distributed on multiple machines we would also have to specify the master host when starting the slaves (this is not needed when running Locust distributed on a single machine, since the master host defaults to 127.0.0.1):

```
$ locust -f locust_files/my_locust_file.py --slave --master-host=192.168.0.100
```

Parameters can also be set in a [config file](#) (locust.conf or ~/.locust.conf) or in env vars, prefixed by `LOCUST_`

For example: (this will do the same thing as the previous command)

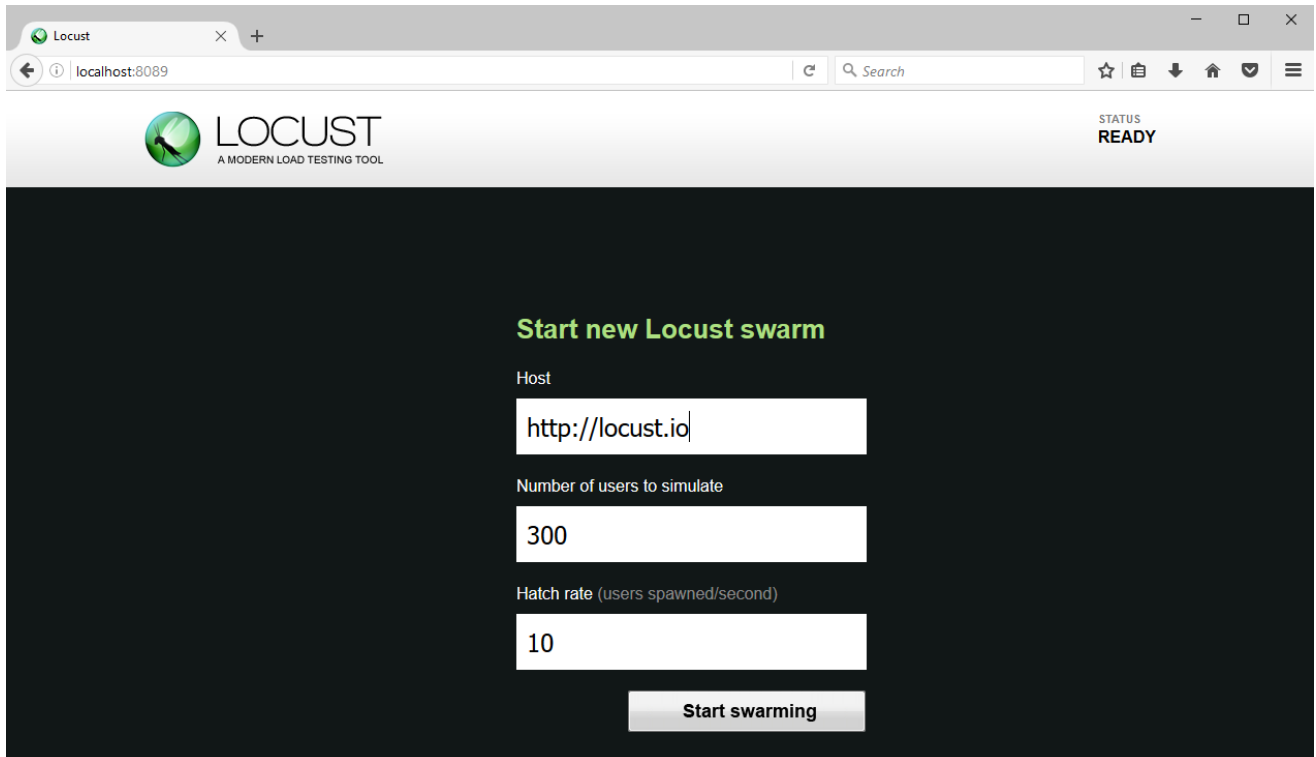
```
$ LOCUST_MASTER_HOST=192.168.0.100 locust
```

### Note

To see all available options type: `locust --help`

## Open up Locust's web interface

Once you've started Locust using one of the above command lines, you should open up a browser and point it to <http://127.0.0.1:8089> (if you are running Locust locally). Then you should be greeted with something like this:



The screenshot shows a web browser window with the title 'Locust'. The address bar shows 'localhost:8089'. The page header features the Locust logo (a green circle with a black locust) and the text 'LOCUST A MODERN LOAD TESTING TOOL'. On the right, it says 'STATUS READY'. The main content area has a dark background and a green heading 'Start new Locust swarm'. Below this, there are three input fields: 'Host' with the value 'http://locust.io', 'Number of users to simulate' with the value '300', and 'Hatch rate (users spawned/second)' with the value '10'. At the bottom right, there is a button labeled 'Start swarming'.

Locust

localhost:8089

LOCUST  
A MODERN LOAD TESTING TOOL

STATUS  
READY

### Start new Locust swarm

Host

Number of users to simulate

Hatch rate (users spawned/second)

Start swarming