Locust Python load testing

CONCEPTS

A locustfile is plain Python: each User class is one simulated visitor running weighted tasks.

Locustfile User + tasks
↓ `locust`
Master spawns, aggregates
↓ `--worker`
Workers run the users
↓ `HTTP`
Target your system

Only load test systems you own or have written permission to test.

RUNNING

`locust`
Start the web UI (port 8089) using locustfile.py

`locust -f` *<file>*
Use a specific locustfile

`locust -H` *<url>*
Target host for all requests

Defaults can live in locust.conf, ~/.locust.conf or pyproject.toml ([tool.locust]). Env vars use LOCUST_<OPTION>, flags win over both.

HEADLESS

`locust --headless -u` *<n>* **`-r`** *<rate>* **`-t`** *<time>*
Fixed run without UI

`-u / --users` *<n>*
Peak number of simulated users

`-r / --spawn-rate` *<n>*
Users started per second

`-t / --run-time` *<time>*
Stop after this long

`--exit-code-on-error` *<code>*
Exit code when requests failed

REPORTS

`--csv` *<prefix>*
Write <prefix>_stats.csv and failures/ exceptions CSVs

`--html` *<file>*
Write an HTML report at the end

DISTRIBUTED

`locust --master`
Start the master (no users run on it)

`locust --worker`
Start a worker, master on localhost

`locust --worker --master-host` *<host>*
Point a worker at a remote master

`--expect-workers` *<n>*
Headless master waits for n workers

LOCUSTFILE

Minimal HttpUser

```
from locust import HttpUser, task,
between

class ShopUser(HttpUser):
    wait_time = between(1, 5)

    @task(3)
    def home(self):
        self.client.get("/")

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

`@task(`*<weight>***`)`**
Pick this task proportionally more often

`wait_time = between(`*<a>***`, `***<b>***`)`**
Random pause after each task

`def on_start(self):`
Per-user setup such as login

`self.client.get(`*<path>***`, name=`***<label>***`)`**
Group dynamic URLs in the stats