Locust Python load testing

START HERE

`locust -f <file>`
Start with web UI

`locust --headless -u 50 -r 5 -t 2m`
Run without UI

`locust -H <url>`
Set target host

`locust --headless --csv <prefix>`
Save CSV stats

`locust --html <file>`
Write HTML report

`locust --master`
Coordinate workers

`locust --worker`
Generate load

`locust -T <tag>`
Run tagged tasks only

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

CONCEPTS

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

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

Python is single-core per process: use workers (or --processes) to use more CPUs.

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

`locust --web-port <port>`
Change the web UI port

`locust -f <a.py>,<b.py>`
Combine several locustfiles

`locust <UserClass>`
Run only the named user classes

`locust --config <file>`
Read options from a config file

`locust -L DEBUG`
Set the log level

`locust --logfile <file>`
Log to a file instead of stderr

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
-t takes 90, 90s, 20m or 1h30m. The timer starts at the beginning of the test, not after ramp-up.

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

HEADLESS (cont.)

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

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

`-s / --stop-timeout <time>`
Let running tasks finish on shutdown

`--exit-code-on-error <code>`
Exit code when requests failed
By default Locust exits with 1 if any request failed.

`--reset-stats`
Clear stats after ramp-up ends
In distributed mode set it on master and workers.

`--json`
Print final stats as JSON

`--json-file <prefix>`
Write JSON stats to a file

In headless mode press w / W to add 1 / 10 users and s / S to remove them.

REPORTS

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

`--csv-full-history`
Add a row per interval to _stats_history.csv
Needs --csv.

`--html <file>`
Write an HTML report at the end
{u}, {r} and {t} in the name become users, spawn rate and run time.

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

DISTRIBUTED (cont.)

`--master-bind-port <port>`
Port the master listens on (5557)

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

`locust --processes <n>`
Fork master plus n workers locally
Uses fork(), so not on Windows. -1 means one per logical core.

`locust --worker --processes <n>`
Fork only n workers on this machine

Copy the same locustfile to every worker. Run the master headless with --expect-workers for CI.

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

`wait_time = constant(<s>)`
Fixed pause after each task

`wait_time = constant_pacing(<s>)`
Run a task at most once per interval

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

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

LOCUSTFILE (cont.)

`self.client.post(<path>, json=<obj>)`
POST a JSON body

`catch_response=True`
Decide pass/fail from the response

`@tag(<name>)`
Label a task for --tags filtering

`-T <tag> / -E <tag>`
Include / exclude tagged tasks
Exclusion wins when both match.

`class <Name>(FastHttpUser)`
Faster HTTP client for heavy load

`class <Name>(TaskSet)`
Group tasks into a nested flow

`abstract = True`
Mark a base class that is not run

`@events.test_start.add_listener`
Hook code in at test start

`class <Name>(LoadTestShape)`
Script the user curve over time
tick() returns (users, spawn_rate), or None to stop.

RECIPES

Check a response body

```
with self.client.get("/",
catch_response=True) as r:
    if "Welcome" not in r.text:
        r.failure("missing
welcome text")
```

CI smoke run that fails the build

```
locust -f locustfile.py -H
https://staging.example.com \
  --headless -u 20 -r 5 -t 1m --
csv out/smoke --html out/
smoke.html
# exit code 1 if any request
failed
```

Distributed on one machine

```
locust -f locustfile.py --
headless --processes 4 \
  -u 400 -r 20 -t 5m -H https://
staging.example.com
```

Distributed across machines

```
# on the master
locust -f locustfile.py --master
--headless --expect-workers 3 -u
900 -r 30 -t 10m
# on each worker
locust -f locustfile.py --worker
--master-host <master-ip>
```