# Gunicorn Python app server

**START HERE**

`gunicorn app:app`
Serve a WSGI app

`gunicorn -w 4 app:app`
Four worker processes

`gunicorn -b 0.0.0.0:8000 app:app`
Choose the address

`gunicorn --reload app:app`
Restart on code change

`gunicorn -c gunicorn.conf.py app:app`
Use a config file

`gunicorn --access-logfile - app:app`
Log requests to stdout

`gunicorn -t 60 app:app`
Raise worker timeout

`kill -HUP <pid>`
Reload workers

**Client** browser → HTTP → **nginx** TLS, static → proxy_pass → **Master** gunicorn, no requests → fork → **Workers** run your app

## CONCEPTS

app:app means module:variable. Gunicorn serves one app per process; put nginx in front for TLS and static files.

The master only manages workers; it never handles requests. A worker silent past the timeout is killed and replaced.

Rule of thumb for sync workers: 2 x cores + 1. Threads or async workers help when requests wait on I/O.

## RUN

`gunicorn <module>:<app>`
Serve a WSGI app

`gunicorn '<module>:<factory>()'`
Call an app factory
*Quote it so the shell keeps the parentheses.*

`gunicorn -w <n> <module>:<app>`
Set worker count

`gunicorn -b <host>:<port> <module>:<app>`
Bind a host and port

`gunicorn -b unix:<path> <module>:<app>`
Bind a Unix socket

`gunicorn <project>.wsgi`
Run a Django project

`gunicorn --reload <module>:<app>`
Auto-restart on code change
*Development only.*

`gunicorn --check-config <module>:<app>`
Check config, then exit

`gunicorn --print-config <module>:<app>`
Show resolved settings

`gunicorn --chdir <dir> <module>:<app>`
Change directory first

`gunicorn --pythonpath <dirs> <module>:<app>`
Add to the Python path

`gunicorn -e <KEY>=<value> <module>:<app>`
Set an environment variable

`gunicorn --paste <file.ini>`
Load a PasteDeploy config

## RUN (cont.)

`gunicorn -v`
Print the version

`GUNICORN_CMD_ARGS="--bind=:8000 -w 4"`
Settings from the environment

## WORKERS

`-k sync`
Default: one request per worker

`-k gthread --threads <n>`
Threaded workers

`-k gevent`
Async workers via gevent
*Needs gevent installed (gunicorn[gevent]).*

`-k asgi`
Run an ASGI app
*Built into recent releases (26.x); older versions need uvicorn's worker class.*

`--worker-connections <n>`
Clients per async worker

`--max-requests <n>`
Recycle a worker after n requests

`--max-requests-jitter <n>`
Stagger those restarts

`-t <sec>` / `--timeout <sec>`
Kill silent workers after n seconds

`--graceful-timeout <sec>`
Grace period on restart

`--preload`
Load app before forking
*Saves memory, but code changes need a full restart, not HUP.*

`--worker-tmp-dir /dev/shm`
Heartbeat files in RAM
*Fixes spurious timeouts in containers.*

`-u <user> -g <group>`
Drop privileges

`-p <file>`
Write a PID file

### Pick a worker

| Class | Fits |
|---|---|
| `sync` | simple CPU-bound apps |
| `gthread` | mixed or blocking I/O |
| `gevent` | many slow connections |
| `asgi` | async frameworks |

## LOGGING

`--access-logfile -`
Access log to stdout

`--error-logfile -`
Error log to stderr

## LOGGING (cont.)

`--log-level <level>`
debug, info, warning, error

`--access-logformat "<fmt>"`
Custom access log format

`--capture-output`
Send app stdout/stderr to the log

`--log-config <file>`
Use a logging config file

`--statsd-host <host:port>`
Emit StatsD metrics

`-n <name>`
Process name (setproctitle)

## SIGNALS

`kill -HUP <master-pid>`
Reload config, replace workers

`kill -TERM <master-pid>`
Graceful shutdown

`kill -INT <master-pid>`
Quick shutdown
*Same for QUIT.*

`kill -TTIN <master-pid>`
Add one worker

`kill -TTOU <master-pid>`
Remove one worker

`kill -USR1 <master-pid>`
Reopen log files
*Use after log rotation.*

`kill -USR2 <master-pid>`
Start a binary upgrade

`kill -WINCH <master-pid>`
Stop workers (daemon mode)

HUP does not reload code that was loaded with --preload; use USR2 or restart.

**USR2** new master starts
↓ WINCH
**Old workers stop** old master idles
↓ QUIT
**Old master exits** new one serves
Zero-downtime binary upgrade

## CONFIG FILE

### gunicorn.conf.py

```
bind = "127.0.0.1:8000"
workers = 4
worker_class = "gthread"
threads = 2
timeout = 30
accesslog = "-"
errorlog = "-"
preload_app = True
max_requests = 1000
```

## CONFIG FILE (cont.)

Precedence: command line beats config file beats defaults. ./gunicorn.conf.py is read automatically.

`-c python:<module>`
Config from a module

### Hooks in the config

```
def when_ready(server):
    print("ready")
def post_fork(server, worker):
    pass
```

## DEPLOY

### systemd unit

```
[Service]
User=www-data
WorkingDirectory=/srv/app
ExecStart=/srv/app/.venv/bin/
gunicorn -c gunicorn.conf.py
app:app
ExecReload=/bin/kill -s HUP
$MAINPID
Restart=on-failure
```

### nginx in front

```
location / {
    proxy_pass
http://127.0.0.1:8000;
    proxy_set_header Host $host;
    proxy_set_header X-
Forwarded-For $remote_addr;
}
```

`--forwarded-allow-ips <ips>`
Trust proxy headers from these IPs
*Needed when the proxy is not on localhost.*

`--keyfile <key> --certfile <crt>`
Serve HTTPS directly

## RECIPES

### Production-ish start

```
gunicorn -w 4 -k gthread --
threads 2 \
  -b unix:/run/app.sock \
  --access-logfile - --max-
requests 1000 \
  app:app
```

### Zero-downtime reload

```
kill -HUP $(cat app.pid)
# new code only if not --preload
```

### Scale workers live

```
kill -TTIN $(cat app.pid)   # +1
kill -TTOU $(cat app.pid)   # -1
```

### Debug a hanging app

```
gunicorn --log-level debug -w 1
-t 0 \
  --access-logfile - app:app
```