

Server Users, Groups, and How Your Sites Are Served (Nginx + Gunicorn + Django)

This document explains **which Linux user/group runs which part of your web stack**, how your **static (vanilla HTML) site** and your **Django site** are served, and the **commands to inspect how everything connects**.

1) The Cast: Users and Groups

Users you've seen

- **root**
 - The admin/superuser.
 - Has unlimited permissions.
- **www-data**
 - Standard system user for **Nginx** worker processes.
 - Low-privilege, designed for web serving.
- **django-user**
 - Your dedicated app user.
 - Runs **Gunicorn** (and therefore **Django**).

Groups you've seen

- **www-data** (group)
 - Used to grant Nginx workers access to shared resources (commonly the Gunicorn socket).
- **sudo**
 - Lets a user run admin commands via `sudo`.
- **django-user** (group)
 - Typically the user's default primary group.

Key idea: **User** = "who" the process runs as. **Group** = additional permission bucket.

2) What Runs as What (Your Actual Server)

From your command outputs:

Nginx

- **Master process:** runs as `root`
- **Worker process(es):** run as `www-data`

Why: - `root` is needed briefly to bind privileged ports **80/443** and manage workers. - Workers drop privileges to `www-data` and do the real request handling.

Gunicorn + Django

- Gunicorn runs as `django-user` (master + workers)
- Django code runs **inside Gunicorn**, so it also runs as `django-user`

Your `gunicorn.service`: - `User=django-user` - `Group=www-data`

Meaning: - Process user = `django-user` - Process group = `www-data` (useful for socket sharing)

3) How Each Site Is Served

A) Static / Vanilla HTML site

Request flow 1. Browser hits your domain on port 80/443 2. Nginx worker (user: `www-data`) reads files from the configured `root` directory 3. Nginx returns HTML/CSS/JS/images directly

Who serves it? - Nginx worker → `www-data`

Permissions principle - The directory containing the HTML must be readable by `www-data`.

B) Django site (dynamic pages)

Request flow 1. Browser hits your Django domain on port 80/443 2. Nginx worker (user: `www-data`) receives the request 3. Nginx proxies the request to Gunicorn via a **Unix socket** (e.g. `/run/gunicorn.sock`) 4. Gunicorn (user: `django-user`) receives the request and calls Django (`baseball_draft.wsgi:application`) 5. Django generates a response 6. Response goes back through Gunicorn → socket → Nginx → browser

Who serves it? - Nginx serves as the public front door (`www-data`) - Django is executed by Gunicorn (`django-user`)

Static files for Django - `/static/` and `/media/` are usually served by Nginx (`www-data`), not by Django.

4) The “Handshake” Between Nginx and Gunicorn

The key connection is the **Unix socket**.

To work correctly, these must match:

1) Gunicorn bind: - `--bind unix:/run/gunicorn.sock`

2) systemd socket (if used): - `ListenStream=/run/gunicorn.sock`

3) Nginx proxy: - `proxy_pass http://unix:/run/gunicorn.sock;`

And permissions must allow: - `www-data` (Nginx worker) can connect to the socket - `django-user` (Gunicorn) can accept on the socket

5) Commands to See How Everything Relates

A) See who Nginx is running as

```
ps aux | grep nginx
```

What to look for: - `nginx: master process` → `root` - `nginx: worker process` → `www-data`

B) See who Gunicorn (and Django) is running as

```
ps aux | grep gunicorn
```

What to look for: - first column should show `django-user` for gunicorn processes

C) Confirm the user/group for a specific Gunicorn PID

(Use a PID from `ps aux | grep gunicorn`)

```
ps -p <PID> -o user,group,cmd
```

Expected: - `USER` = `django-user` - `GROUP` = `www-data` (because of `Group=www-data` in the service)

D) See which groups a user belongs to

```
groups django-user
```

(You currently have: `django-user : django-user sudo`)

E) See socket ownership and permissions

```
ls -l /run/gunicorn.sock
```

You want the group to be `www-data` so Nginx can connect.

F) See the Nginx site config that serves each domain

```
ls -l /etc/nginx/sites-enabled
```

Then open a specific file:

```
sudo nano /etc/nginx/sites-enabled/<site-name>
```

What to look for: - `server_name ...;` - `root ...;` (static site) - `proxy_pass http://unix:/run/gunicorn.sock;` (Django site)

G) Find which config file defines the Nginx worker user

```
sudo grep -R "^user" -n /etc/nginx/nginx.conf
```

Typically: - `user www-data;`

H) See systemd units for Gunicorn

```
systemctl status gunicorn  
systemctl status gunicorn.socket
```

And view their contents:

```
sudo systemctl cat gunicorn  
sudo systemctl cat gunicorn.socket
```

I) Prove Gunicorn is responding via the socket (direct test)

```
curl --unix-socket /run/gunicorn.sock http://localhost/
```

If you get HTML back, Gunicorn+Django are responding.

6) Quick Summary Table

Component	Purpose	Process User	Process Group	Notes
Nginx master	binds ports, manages workers	root	root	master isn't the main request handler
Nginx worker	serves static + proxies	www-data	www-data	serves vanilla HTML directly
Gunicorn	app server for Django	django-user	www-data	receives proxied requests via socket
Django	your web app	django-user	www-data	executed inside Gunicorn

7) Most Common “Where it breaks” Checklist

- **502 Bad Gateway**
 - Nginx can't reach the socket (`proxy_pass` mismatch)
 - Socket permissions don't allow `www-data` to connect
 - Gunicorn isn't running or crashed
- **Static files not loading**
 - `location /static/` wrong
 - static directory permissions not readable by `www-data`

8) Your Current Notable Detail

- `django-user` is **not** currently a member of `www-data` group.
- That's not automatically “broken,” because systemd can run the process with `Group=www-data`.
- But adding membership can make day-to-day permissions and deployments smoother.

Command to add (optional, common):

```
sudo usermod -aG www-data django-user
```

(Then log out/in for new group membership to apply in your shell sessions.)