CSC309H1S

Programming on the Web

Winter 2023

Lecture 12: Web Deployment

Instructor: Kuei (Jack) Sun

Department of Computer Science
University of Toronto

v1.01

Introduction

- So far
  - Frontend development
    - HTML/CSS
    - Client-side scripting with JavaScript
    - React
  - Backend development
    - Django
  - Everything currently running on the *local computer*!
- This lecture
  - Deployment of a web application
  - DevOps

Development vs. Production

Development	Production
Uses lightweight database, e.g. SQLite	Uses real database, e.g. MySQL, MongoDB
Runs a development server	Runs a real webserver, e.g., Nginx
Hosts on local machine	Hosts on public machine or cloud platform
Hosts on a local IP address	Hosts on a static IP address
Does not have a domain name	Has a domain name
Upon error, shows stack trace	Upon error, returns 500 or 404
Security is not a concern	Needs to be secure and robust
Cannot not handle high traffic	Can handle high traffic

IP Address

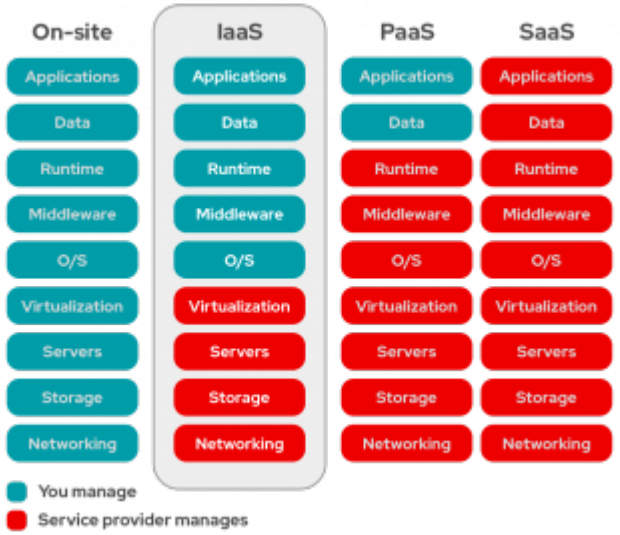
- Most IPv4 addresses have almost been used up
- Transition to IPv6 has been slow (~34% globally in 2022)
- Static IP address
  - Fixed IP address for the machine
  - Does not change over time, even if you power off your machine
- Dynamic IP address
  - IP address assigned by DHCP server
  - Can change the next time you connect to the Internet
- Production server typically uses static IP address(es)

Domain Name

- Your website likely needs a domain name
  - Otherwise, users must use the IP address directly
- Domain Name Registrar
  - Handles reservation of domain names
  - Assigns IP addresses to those domain names
  - Example
    - GoDaddy
    - NameCheap
- You will need to buy a domain name from a registrar
  - Price varies depending on popularity and top-level domain
    - E.g., `.com` is the most popular top-level domain

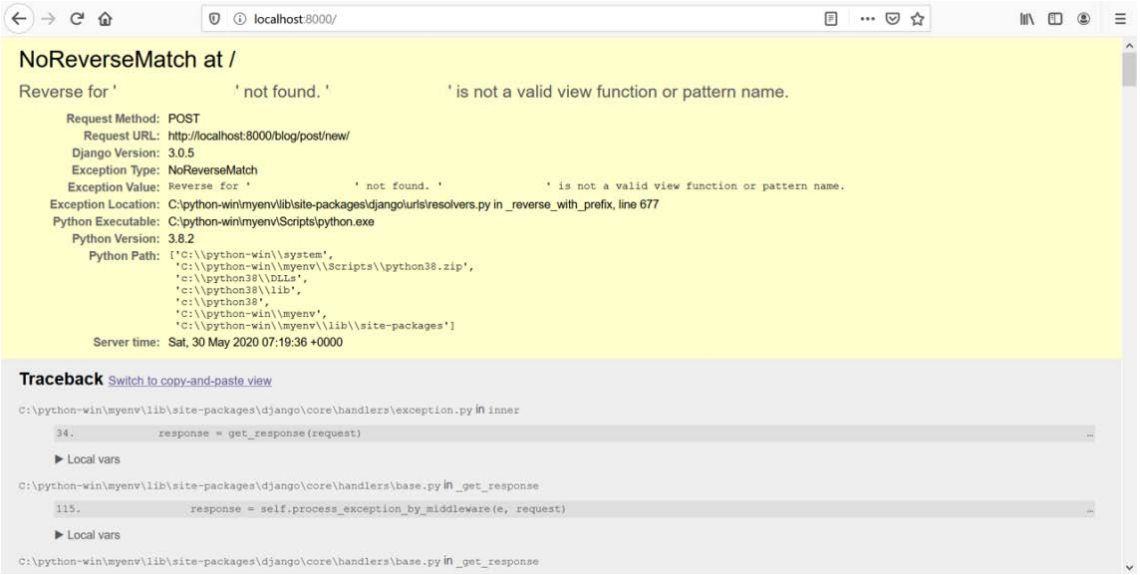
Web Hosting

- Various options available
- Dedicated Hosting
  - Entire physical server to your website
  - Poor utilization, scalability, and availability
- Cloud hosting
  - Running application using combined computer resource
  - IaaS (Infrastructure-as-a-service):
    - User manages OS and above
  - PaaS (Platform-as-a-service):
    - User manages application and data
  - SaaS (Software-as-a-service)



Error Handling

- During development, Django provides nice diagnostic messages



- During production, you will only see 500 or 404 errors

Not Found

The requested URL was not found on this server.

Apache/2.4.29 (Ubuntu) Server at cs.toronto.edu Port 80

Deploying Django Project

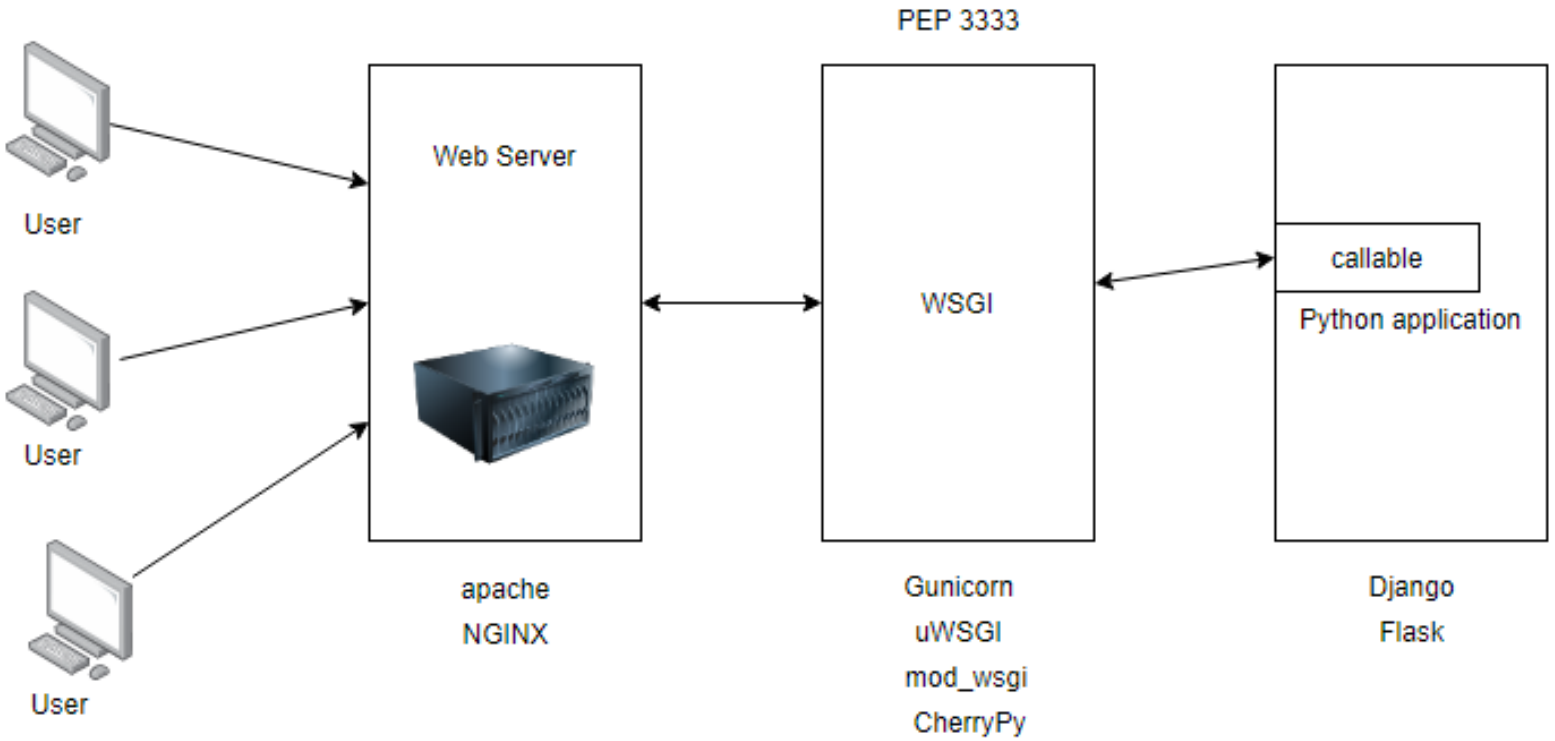
Deployment Options

- You can directly run development server on http port 80
  - `sudo python3 manage.py runserver 0.0.0.0:80`
    - sudo is required because using port 80 required root privilege
  - Highly not recommended
- Gunicorn (Green Unicorn)
  - A fast and lightweight WSGI HTTP server
  - Installation
    - `sudo pip3 install gunicorn`
  - Start in terminal
    - In the Django project root folder, run:
      - `gunicorn --bind 0.0.0.0:80 <project>.wsgi`
    - Does not serve static file!

Note: `<project>.wsgi` is name of a *new* file

WSGI

- Requests are forwarded to your Python application via WSGI
- Works for any webserver and any Python backend framework



Source: https://medium.com/analytics-vidhya/what-is-wsgi-web-server-gateway-interface-ed2d290449e

Apache Webserver

- Django's development server
  - Not meant to handle high loads, withstand attacks, etc
- Installation on Ubunbu
  - Apache webserver
    - `sudo apt-get install apache2 apache2-dev apache2-utils`
  - mod_wsgi (for Apache)
    - `sudo apt-get install libapache2-mod-wsgi-py3`
  - mod-wsgi (for Django)
    - `sudo pip3 install mod-wsgi`
  - Copy or entire project folder into `/var/www/`
    - You may need to change owner/group to `www-data`
      - `sudo chown –R www-data:www-data /var/www/<project>`

Set up Apache conf file

- Create `/etc/apache2/sites-available/<project>.conf`

```
<VirtualHost *:80>
    ServerAdmin admin@your-domain.com
    ServerName your-domain.com
    DocumentRoot /var/www/django_project/
    ErrorLog ${APACHE_LOG_DIR}/your-domain.com_error.log
    CustomLog ${APACHE_LOG_DIR}/your-domain.com_access.log combined

    Alias /static /var/www/django_project/static
    <Directory /var/www/django_project/static>
        Require all granted
    </Directory>
    <Directory /var/www/django_project/django_app>
        <Files wsgi.py>
           Require all granted
        </Files>
    </Directory>

    WSGIDaemonProcess django_app python-path=/var/www/django_project python-
home=/var/www/django_project/venv
    WSGIProcessGroup django_app
    WSGIScriptAlias / /var/www/django_project/django_app/wsgi.py
</VirtualHost>
```

Run these afterwards:

```
a2ensite project.conf
systemctl reload apache2
```

Set up Django

- Production settings (`settings.py`)
  - DEBUG
    - Should set to False
  - SECRET_KEY
    - Use a secure secret key instead
  - ALLOWED_HOSTS
    - Should add your domain name
  - DATABASE
    - You should probably not use SQLite for production purposes
    - https://docs.djangoproject.com/en/4.1/ref/databases/
- Do not push settings.py into repository!
  - Can be a security leak if you do because database password is stored

Production Settings

- DJANGO_SETTINGS_MODULE
  - Django loads settings from this environment variable
  - Can create another file that imports from settings.py and override some options
    - E.g., `export DJANGO_SETTINGS_MODULE=project.production_settings`
- `if DEBUG:`
  - Can separate debug versus production settings
- .env file
  - Load settings from an environment file
  - One for local, one for production
  - Load it on startup
    - python-dotenv package

```
DATABASES = {
    'default': {
        'ENGINE': os.environ['DB_ENGINE'],
        'NAME': os.environ['DB_NAME'],
        'USER': os.environ['DB_USER'],
        'PASSWORD': os.environ['DB_PASSWORD'],
        'HOST': os.environ['DB_HOST'],
        'PORT': os.environ['DB_PORT'],
    }
}
```

Static Files

- Static files are file/directory access granted to the webserver
- In Django project, they are scattered in many places
  - app/static folders
  - Global static folder
  - From Django contrib package, e.g., admin panel
- Django can *collect* all of them into STATIC_ROOT folder
  - Required for security and performance reasons
  - Typically served by the webserver
    - Should *not* go through URL dispatcher!
  - Command:
    - `python3 manage.py collectstatic`

Advanced Setup

- Combine multiple webservers
  - Each has a dedicated task
    - E.g., serving dynamic content, static files, etc.
  - Can exist on a different physical machine or in a CDN
- Gunicorn
  - Currently stops when you close the terminal
    - You can start it using nohup, which will not shutdown when terminal closes
    - However, it does not restart if machine reboots
  - Solution: make it a service
    - Runs forever
    - Restarts upon error
    - Runs on startup

Service

- Linux has a tool to register and monitor your services
- Create new file under the following directory
  - `/etc/systemd/system/<service_name>.service`

```
[Unit]
Description=first_projet gunicorn daemon
After=network.target

[Service]
User=ubuntu
WorkingDirectory=/home/ubuntu/first_project
ExecStart=/home/ubuntu/first_project/venv/bin/gunicorn \
                --access-logfile - \
                --workers 3 \
                --bind unix:/home/ubuntu/first_project/first_project.sock \
                first_project.wsgi:application

[Install]
WantedBy=multi-user.target
```

Service Cont.

- Manage your service via these commands:
  - `sudo service <name> restart`
  - `sudo service <name> status`
  - `sudo service <name> stop`
- If you change the service file, you might need to reload it
  - `sudo systemctl daemon-reload`
- Note that it binds to a socket, not 0.0.0.0:8000
  - We want a real webserver to serve our application on port 80
  - We cannot let gunicorn take port 80. Why?
- A real webserver forwards dynamic requests to gunicorn
  - Gunicorn services the backend project through a local socket

Deploy with Nginx

- `/etc/nginx/sites-available/<project_name>`
  - Create your config file here
- Make a symbolic link to that file
  - At `/etc/nginx/sites-enabled/`
- Restart Nginx!

/etc/nginx/sites-available/myproject

```
server {
    listen 80;
    server_name server_domain_or_IP;

    location = /favicon.ico { access_log off; log_not_found off; }
    location /static/ {
        root /home/sammy/myproject;
    }

    location / {
        include proxy_params;
        proxy_pass http://unix:/home/sammy/myproject/myproject.sock;
    }
}
```

https://www.digitalocean.com/community/tutorials/how-to-set-up-django-with-postgres-nginx-and-gunicorn-on-ubuntu-22-04

Deploying React Project

Deploy React Project

- Much easier than backend servers

1. Build your React project
   - `npm run build`
2. Configure webserver to serve the appropriate files

- Nginx example
  - Just route all requests to the build folder
  - Then, restart Nginx

```
server {
    listen 80;
    server_name mysite.com www.mysite.com
    root /home/ubuntu/app-deploy/build;
    location / {
        try_files $uri /index.html;
    }
}
```

- Further reading
  - https://enrico-portolan.medium.com/how-to-host-react-app-with-nginx-634ad0c8425a