**Deployment**

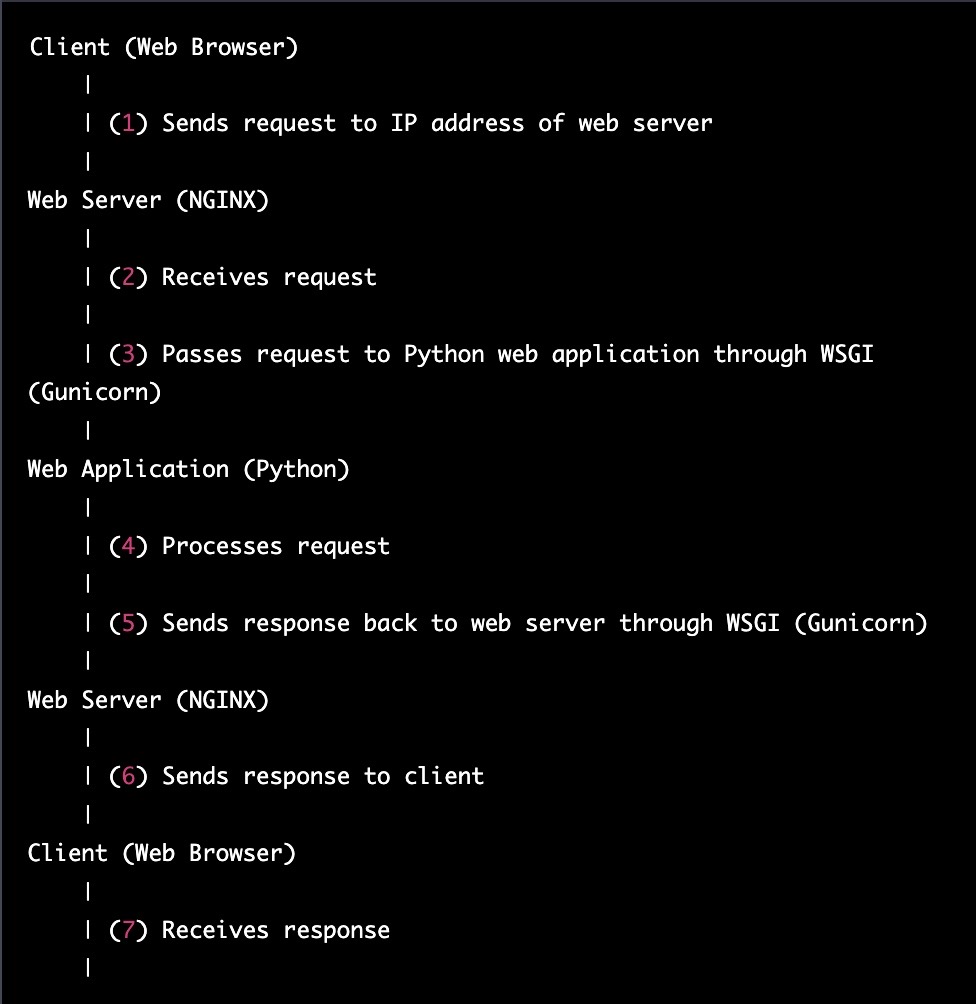
- Deploying apps to the cloud. What this means is setting up your application on a remote machine that can be accessed by browsers sending requests to your server.

- There are many web server hosting choices. We will be using **Amazon Web Services (AWS)**. Amazon has servers all over the world consisting of many powerful computers. These computers are connected to the internet, allowing us to rent out a small piece of that server for our use.

- When we deploy, we are setting up a web server on the AWS computer and connecting it to our application. When a client makes a request to our IP address, it is received by the web server. We are using an open source web server called ****NGINX****. NGINX is used for things like web serving, caching and load balancing. Next, NGINX needs to be able to talk to our application. This is where a ****WSGI**** (Web Server Gateway Interface) comes in. WSGI is what Python web apps use to connect to the web server (NGINX). We are using a certain implementation of a WSGI called ****Gunicorn**** (or Green Unicorn). When we set up Gunicorn, we create a **.sock file**. That .sock file is a UNIX domain socket that plugs Gunicorn into our app and NGINX. Three instances of these application/web server connections will be implemented to handle heavier traffic. With our application plugged into our web server, we are able to handle requests from the client.

**- Simpler explanation of the process:**

1. A client (such as a web browser) makes a request to a web server by sending a request to the server's IP address.
2. The request is received by the web server, which is a program running on a computer (in this case, an AWS computer) that is responsible for receiving and handling requests from clients.
3. The web server communicates with the web application (in this case, a Python application) through a connector called a WSGI (Web Server Gateway Interface). In this particular case, the WSGI implementation being used is called Gunicorn.
4. The web application processes the request and sends a response back to the web server through the WSGI connector.
5. The web server sends the response back to the client.



# Deployment Basics

# - Steps required to deploy a Python project:

##### Your Server

- We are using Amazon Web services to deploy our Python project. That means we have an **Ubuntu server, a Linux operating system**! However, it does not have anything else. Think back to how we had to do a number of installations on our own computers before we could start building Python projects. We will need to do the same for our Ubuntu server.

- The ****PEM key**** we download when we create our server instance is what gives us permission to ****SSH** (Secure Shell)** into our server. Never put your PEM key on GitHub or in any other way share your PEM key. **Keep it Secure!!**

- When we SSH into our server, we create a **pipe** between our computer and our server. We will have a terminal window that is actually a window into our terminal. Keep in mind that this window is no longer your own computer! We are now on our Ubuntu server, and we need to do the same installations as we did on our own computer so that we may run our Python project. For now, **we will only interact with our Ubuntu server through this terminal window**. We will not be able to use VSCode or MySQLWorkbench or any other GUI.

- **Vim commands** may be used in various parts of the process of deploying a Python project on an Ubuntu server using Amazon Web Services (AWS).

##### Your Code

When our Ubuntu server has everything it needs to run our Python project, we need to put our code in our server! To do this, we will use GitHub. **We push our code from our computer onto GitHub, then pull the code from GitHub into our Ubuntu server.**

##### Deploy!

Once our Ubuntu server has everything it needs, we'll set up Nginx and Gunicorn. Follow all the steps very closely and think about what each command is for!

**Servers**

- Server is simply a computer like the one you're using now, but server's sole purpose is to handle requests and send back some sort of response. In order to do so, a server is designed with much more processing speed, storage, and RAM.

**Web Server Components**

Let's look more closely at what's happening on the server side when the client makes a request.

1. **Web Server**- The web server receives a request from the client. The web server asks the logic unit to retrieve some content.
2. **Application**- The logic unit receives requests from the web server. Its job is to run appropriate segments of your code according to the request's instructions. If that code requires information from the database, the logic unit will retrieve that data before organizing it and sending back a response to the web server.
3. **Database**- The Database is simply a file or a container for document storage like your computer's hard-drive. Its sole purpose is to store files, update files, and retrieve files according to instructions received from the interpreter.

**Process Managers**

- Green Unicorn as a process manager

- In general, the **process manager** performs the following **roles**:

1. Restarts the app automatically if it crashes
2. Gains insights into runtime performance and resource consumption
3. Modifies settings dynamically to improve performance
4. Load Balancing

- **Green Unicorn (Gunicorn),** performs these roles by acting as an intermediary between incoming requests and your Django app. While you were testing your app on your local machine, it was ok to use Django’s built-in, light-weight testing server, but that server wasn’t designed to be used in a deployment environment.

- Gunicorn, on the other hand, is designed to **handle incoming requests that will need to be routed to the interpreter.** This includes any route you've added to urls.py.

- Gunicorn needs a way to communicate with our app. When creating a new Django project you may have noticed that a document called wsgi.py is automatically generated. What you need to know is that your **wsgi.py file is the glue that connects the Gunicorn server to your Django app.** It’s how Gunicorn knows where to look for all that good code you wrote!

~~- Chances are that when you deploy an app you’ve been developing in a local environment, you need to use some kind of process manager, regardless of the technology stack. Some common combinations include Unicorn for Rails, pm2 for NodeJS, and Apache for PHP.~~

- There are three basic functions you will need to set up when you add Gunicorn to your project. To add this functionality, Gunicorn needs some instructions on how to do the following:

1. Which socket to connect to
2. What to do if a process fails
3. How many workers to set up

~~- Armed with these instructions, Gunicorn takes over the job of ensuring that your app keeps running if an error occurs, directing traffic to the appropriate port, and running concurrent processes. In short, Gunicorn makes sure your deployment server runs more smoothly.~~