COSC 419 – Topics in Computer Science

Fall 2020

Recap: Last Lecture

- Last lecture, we took our first look at what it means to do ***full stack*** web development – many different hats to wear
- We discussed the major components of a web development:
  - The operating system (CentOS)
  - The web server (Apache, NGINX)
  - The front end (HTML, CSS, JS)
  - The back end (Python, PHP, frameworks)
  - The database (MySQL, MariaDB, SQLite)

Recap: Lab 1

- In our first lab, we got our first look at the web servers that we'll be using, running the **CentOS** operating system
- We installed our first major dependencies:
  - We installed and enabled our **Apache** web server
  - We installed our first backend language, **Python**
- I had you write a very basic Python application which dynamically changed an HTML page on a timer, which was served by Apache

Working on the Backend

- We are now at a crossroads – we've installed our backend language, but how are we going to use it?
- We could write our entire backend from scratch in Python, implementing all the methods and functions we would require to communicate with a client via the web server
  - For very niche uses, this is sometimes the only option
- Or, we could opt to use an already existing **framework** to develop our backend logic
  - The more common choice for the modern web developer

Why Use A Framework?

- There is no reason to reinvent the wheel: frameworks provide ready-to-use implementations for many common web server functions i.e. handling cookies, dealing with POST request data, etc
- Not only does this save time, but makes it less likely that exploitable bugs slip into your software

The Flask Framework

- The first framework that we'll be looking at in this course is the **Flask** framework for Python
- Flask is often described as a **microframework**, as it lacks some of the common features such as form validation, middleware, and database handling
  - In my opinion, not necessarily an important distinction
- However, being Python based, all of these features can be found in existing libraries and extensions which can be easily integrated into Flask

Why Flask?

- Sometimes, less is more: although Flask may lack some features found in more comprehensive frameworks, it opens the door for very **free-form** web development
- It uses a very simple structure and implementation compared to full-featured frameworks like Laravel – it is entirely possible to write your entire web application in a single Python file
- Python is generally easier to comprehend than PHP, which makes Flask applications easy to understand (albeit with some formatting quirks – **whitespace is important!**)
- Licensed under **BSD** license (free and relatively open)

Integrating Flask with Apache: WSGI

- The Flask framework cannot simply interface directly with Apache; we need some sort of handler between Apache and Python
- This handler is the **Web Server Gateway Interface** (**WSGI**) – a set of conventions that handle communication between Python applications and frameworks (like Flask), and web servers (like Apache and NGINX)
- WSGI requires a little configuration, but is fairly straightforward – a handful of config files are all that's required

Installing WSGI on CentOS

```
================================================================================
 Package                Architecture Version             Repository       Size
================================================================================
Installing:
 python3-mod_wsgi       x86_64       4.6.4-4.el8         AppStream       2.5 M

Transaction Summary
================================================================================
```

- To setup WSGI, we just need to install the appropriate module for Apache (hint: if you see **mod**_<package name>, it's probably an Apache module):

```
sudo yum install python3-mod_wsgi
```

- Note that mod_wsgi is also thing for Python 2 – make sure that you specify the **python3** version

Configuring WSGI

```
<VirtualHost *>
    WSGIDaemonProcess myapp user=apache group=apache threads=5
    WSGIScriptAlias / /var/www/html/myapp.wsgi

    <Directory /var/www/html>
        WSGIProcessGroup myapp
        WSGIApplicationGroup %{GLOBAL}
        Order deny,allow
        Allow from all
    </Directory>
</VirtualHost>
```

- To start, we'll need to navigate to the following folder:

  `/etc/httpd/conf.d`

- This folder is used for custom configuration (.conf) files for Apache
- We'll create one called **wsgi.conf**, and we'll use it to point Apache to our web application
- Here we specify a new VirtualHost for all traffic (*)

Using WSGI on it's Own

- We don't actually need Flask installed to test out our WSGI installation – it'd designed to work with generic Python applications, not just frameworks
- If you create a **myapp.wsgi** file in **/var/www/html**, the following code will produce a simple 'Hello World' web application

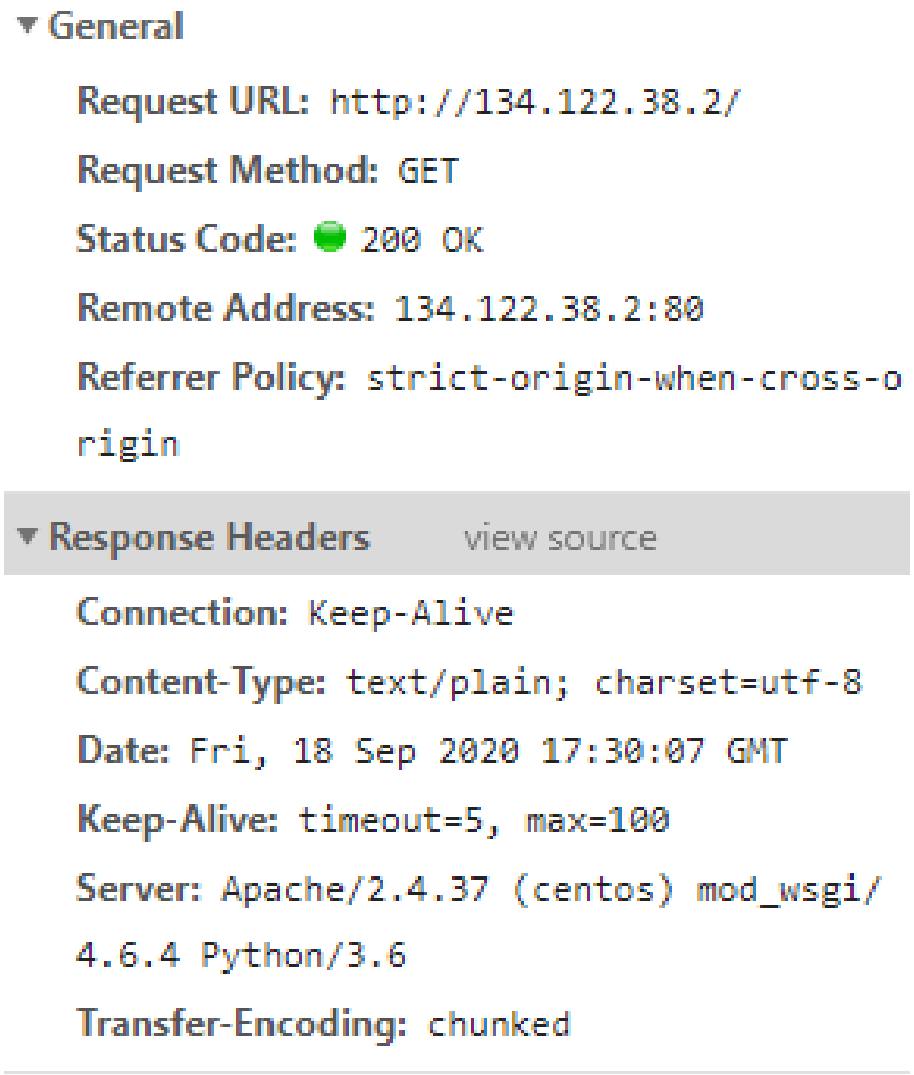
```
def application(environ, start_response):
  status = '200 OK'
  headers = [('Content-type', 'text/plain; charset=utf-8')]
  body = 'Hello World!'.encode('utf-8')
  start_response(status, headers)
  return [body]
```

Using WSGI on it's Own

- After adding our **myapp.wsgi** file, we'll need to quickly reboot Apache:

```
sudo service httpd restart
```

- Going to our server in a web browser allows us to see our "Hello World!" page.
- A quick inspector check shows the server reporting mod_wsgi and Python 3.6 in it's software stack



Bugtesting and Logging

- Before going any further, it's important we talk about bugtesting on our servers
- **When in doubt, check the server logs!** The Apache server log can be found at **/var/log/httpd/error_log**
  - This log is continually being written to, so you'll need to scroll all the way to the bottom to find the latest exceptions/errors
- **About 95% of the errors you encounter in web development will end up in this log file**

Installing Flask

- Now we're just about ready to install Flask
- We'll need to use **pip3** (also known simply as pip) for this task; pip is to Python what yum or apt are to Linux – a package manager
- Pip will download and install Python modules for us; to install Flask all we need to do is run:

```
sudo pip3 install Flask
```

- It will download and install Flask, as well as all the required dependencies such as Jinja and Werkzeug

Changing our WSGI Configuration

- With Flask, we'll need to modify our **myapp.wsgi** file in **/var/www/html** a bit:

```
import sys
sys.path.insert(0, '/var/www/html')
from myapp import app as application
```

- The important thing to note here is that we'll be importing our Flask **application class** from **myapp** – which **isn't** referring to this **myapp.wsgi** file, but a new (not yet created) **myapp.py** file, which will contain our actual web code

Creating Our First Flask App

- Now we can create a new file, **myapp.py** – this will house our actual Flask application
- For a simple “Hello World” app, the following code will suffice:

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello_world():
    return 'Hello, World!'
```

- Notice that we first create an **app** object, which is the object being referenced in our **myapp.wsgi** file

Simple Routing in Flask

- We can define routes using the **@app.route(‘<route>’)** directive
- The directive is followed by a Python **function**, which contains the logic for that particular route
- Routes are analogous to pages in a static (HTML-only) web application: they represent a different URL that can be accessed (potentially with different content)

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello_world():
        return 'Hello, World!'

@app.route('/home')
def home():
        return 'Welcome Home!'

@app.route('/1')
def get_one():
        return '1';
```

Next Week

- Next week, we'll start really digging into what Flask can offer us as a framework:
  - Templating using the Jinja engine
  - Passing data from the backend to templates safely
  - Routing for GET and POST requests, routing with variable URLs
  - Handling session data
- In the lab, we'll be configuring our Flask servers, and then developing our first real web application: a scraper and rudimentary keyword analyzer

Any Questions?