



UWB
HACKS
THE
INTERNET

Flask Fun

Intro to Web Servers

by Thomas Kercheval, Chris Johnston
& The ACM@UWB



Before We Begin

Intended audience:

Beginners (no server dev. exp. needed)

Folks who can read basic Python*

Recommended software:

- Python3
- Git

Scan QR code to find sample code

<https://bit.ly/2OLGuX6>

(<https://github.com/UWB-ACM/flask-barebones>)



*If you don't know Python, that's okay! 2



Before We Begin

As with all of the other presentation materials today, you can download these slides and read more at

<http://docs.uwbhacks.com/>

If you have questions, I'll be happy to answer them at the end of the slides.

Please raise your hand if you have questions.



Before We Begin

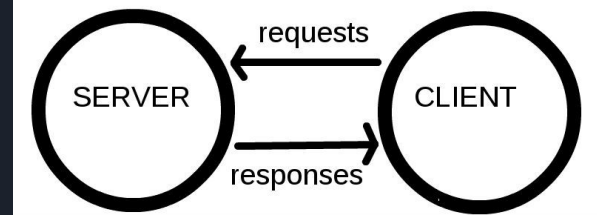
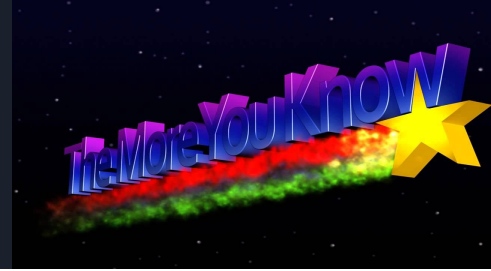
This presentation will focus on Flask, a Python web service microframework. Many of the same concepts exist in other frameworks, available in other languages and tech stacks.

- C#: ASP.NET
- Java: Spring
- C++: CppCMS
- Node.JS: HTTP Module
- Go: net/http
- Python: **Flask**, Django
- And many others that I can't list here

What is a Web Server?

Web servers serve web content!

- “Hosts” HTML & media
 - HTML, CSS, JS, media
 - JSON/XML APIs
- Enables communication among services
- Responds to client requests using HTTP





What is Flask?

Flask is a Web Microframework

- Written in Python
- Simple!
- Efficient (Fast)



Use it to build web applications!



What is Flask?

Why would I want to use a “micro” framework?

<http://flask.pocoo.org/docs/1.0/foreword/#what-does-micro-mean>

“Micro” = simple and extensible.

- Provides only it's own functionality
- Extensible using other libraries
- No point in redoing the work of other libraries.



Routes (API Endpoints)

Routes are hierarchical sections of a website

Full path specifies resource requested from server

URL*	<code>http://flask.pocoo.org/community/poweredby</code>
URL Expansion	<code>http</code> <code>:://flask.pocoo.org</code> <code>/community</code> <code>/poweredby</code>
Annotated URL Parts	<code>PROTOCOL</code> <code>:://HOST_NAME</code> <code>/ROUTE</code> <code>/SUB_ROUTE</code>

*URL – Uniform Resource Locator

<code>PROTOCOL</code>	:: Rules that govern communication
<code>HOST_NAME</code>	:: Name of service (e.g., google.com)
<code>ROUTE</code>	:: Specific page on website/service
<code>SUB_ROUTE</code>	:: Page “deeper” in the hierarchy



HTTP Methods

The HyperText Transfer Protocol (HTTP) has several methods of operation. Here are some of the most common ones, with their expected behavior.

- GET - Fetch specified resource
- POST - Submit information to resource
- DELETE - Deletes specified resource
- ... and more!

HTTP Request format (including URL):

METHOD **PROTOCOL**:://**HOST_NAME**/**ROUTE**/**SUB_ROUTE**

E.g., **GET** **http**:://**127.0.0.1:5000**/**static**/**index.html**

“PaaS” Example Project

We made an example project which contains the code that we are going to refer to.

<https://bit.ly/2OLGuX6>

<https://github.com/UWB-ACM/flask-barebones>





Setup Project



1. Clone project from GitHub

```
git clone https://github.com/UWB-ACM/flask-barebones.git
```

2. Enter project repo

```
cd flask-barebones/
```

3. Install dependencies

```
python3 -m pip install -r requirements.txt
```



Launching Dev Server

1. Enter `src/` directory

```
cd src/
```

2. Launch dev server

```
python3 main.py
```

3. Visit `127.0.0.1:5000/`

```
[01:09:45] tkerchev@tkerchev3481: ~/projects/flask-barebones $ cd src/
[01:09:51] tkerchev@tkerchev3481: ~/projects/flask-barebones/src $ python main.py
* Serving Flask app "main" (lazy loading)
* Environment: production
  WARNING: Do not use the development server in a production environment.
  Use a production WSGI server instead.
* Debug mode: on
* Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
* Restarting with stat
* Debugger is active!
* Debugger PIN: 142-713-257
127.0.0.1 - - [03/Apr/2019 01:10:16] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [03/Apr/2019 01:10:16] "GET /favicon.ico HTTP/1.1" 404 -
```



Live Demo

Try to use this ngrok link for the server running on my laptop:

TODO

Else, use: <http://paas.uwbhacks.com/>



Defining a Route (in Flask)

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

@app.route('/test')
def testing():
    return 'Hello, Flask!'

if __name__ == '__main__':
    app.run(debug=True,
            host='0.0.0.0')
```

Import web microframework

Get WSGI application instance

Define “test” route as a simple hello world

This just returns plain text, not HTML.

Start Dev server, when script is run directly.

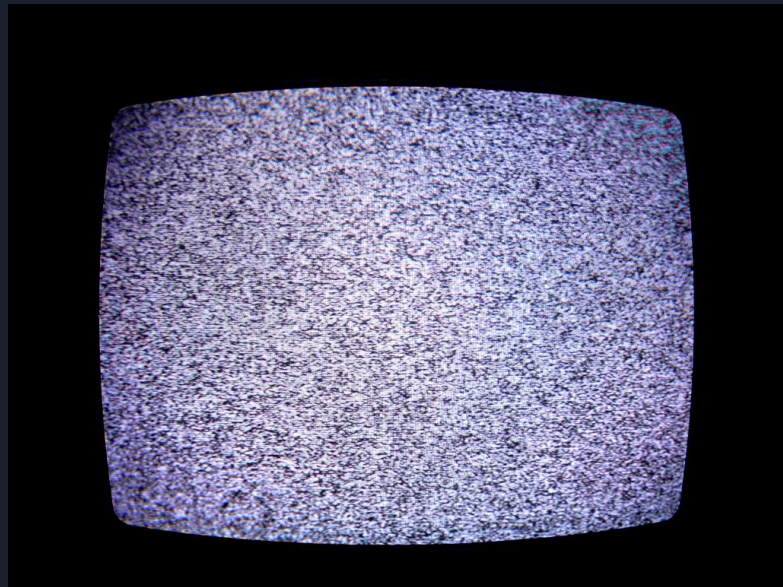
Serving Static Content

Static content...

- Does not change based on behavior
- Sending files to client on request
- HTML, images, videos

Efficient!*

- No additional compute required



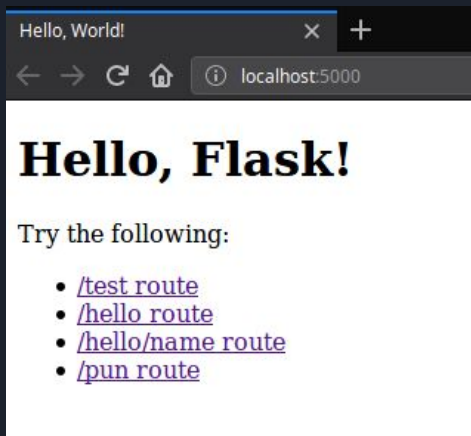
*Static content is best served upstream (e.g., w/Nginx, more later)

Serving Static Content (cont...)

1. Choose a directory to serve from
 - o I chose `src/static/`
2. Create a HTML file
 - o e.g., `src/static/hello.html`
3. Setup a route to serve static content
4. Start the dev server
5. Request resource from browser ->

`src/main.py`

```
@APP.route("/")
def index():
    """
    Responds with the "Hello World" page for the default endpoint.
    """
    return send_from_directory("static", filename="hello.html")
```

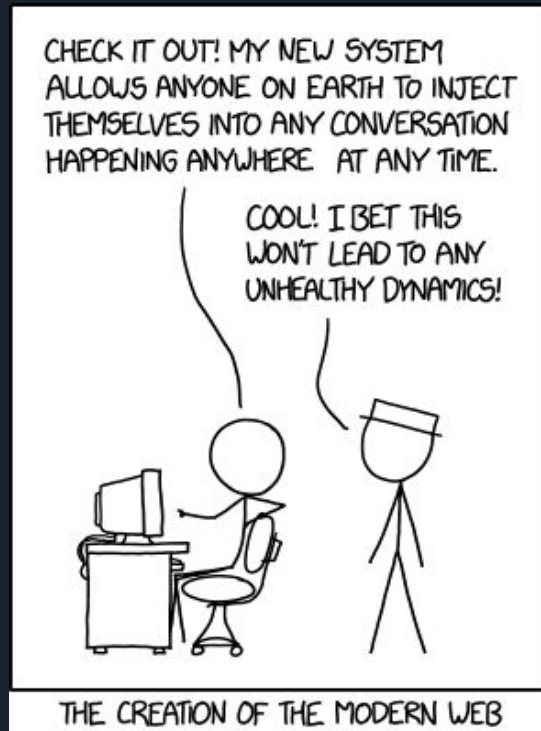


Serving Dynamic Content

Dynamic content changes
based on user behavior!

Examples of dynamic content...

- User account information
- Live weather information
- Content that requires a database call





Serving Dynamic Content (cont...)

HTML Templates in Flask

- Insert `{{ variables }}` into HTML
- Jinja2 - Template Engine
- Rendered at runtime

`src/templates/name.html`

```
<!doctype html>
<html>
  <body>

    <h1>Hello, {{ name }}!</h1>

  </body>
</html>
```

1. Create `templates/` directory
2. Create HTML file, with variable “`{{ name }}`”

Serving Dynamic Content (cont.....)

3. Setup route to accept a parameter
4. Render HTML file
 - Flask will look in `/templates` dir
 - Pass in keyword arguments
 - Keyword matches var name in HTML
5. Start dev server
6. Request resource from browser ->

`src/main.py`

```
@APP.route('/hello')
@APP.route('/hello/<string:name>')
def hello(name = "Anonymous"):
    return render_template('name.html', name=name)
```



Deploying Your Application

Why do I need another server?

- Flask is **NOT** a web server
 - Flask is an app microframework
- Flask's built-in “dev” server is **NOT** meant to scale to production
- Upstream servers typically handle connections and HTTP validation

Static content is best served by
upstream servers





Deploying Your Application

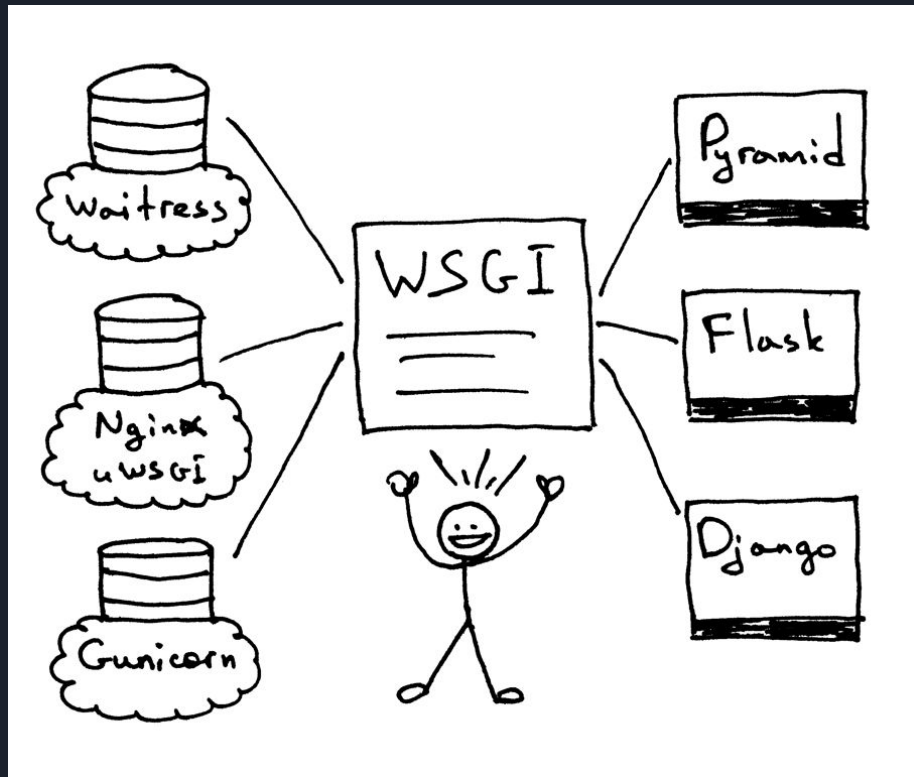
From the flask docs:

While lightweight and easy to use, Flask's built-in server is not suitable for production as it doesn't scale well and by default serves only one request at a time.

WSGI

Web Server Gateway Interface

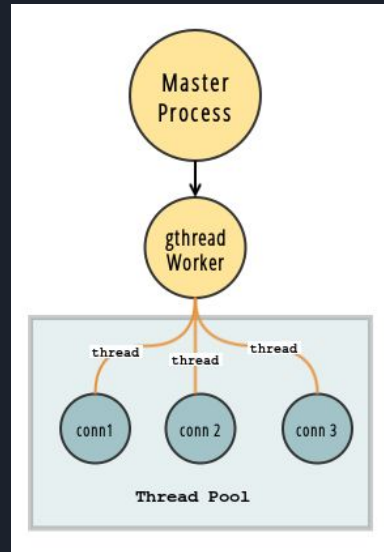
- Interface for upstream servers to talk to Python frameworks
- Enables separation of concerns for app and server devs
- Defined in PEP 3333



Gunicorn Deployment

Gunicorn is a WSGI compliant HTTP server

- Listens for client requests and runs your Flask app
- Helps handle multiple connections
- Pre-fork worker model
 - Thread per worker



Use gunicorn with sample project!

- Run `./gunicorn_start.sh`



gunicorn

Nginx Deployment

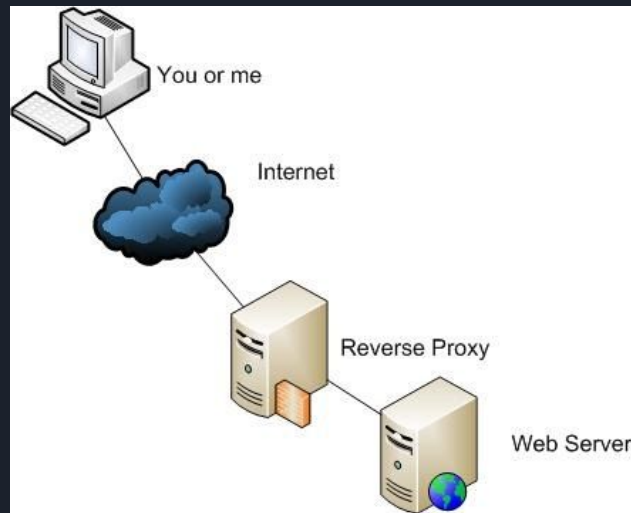


Nginx is a reverse proxy! (yxorp)

- A reverse proxy is an upstream server
- Serves static content VERY efficiently
- Handles HTTP request validation
 - Stops malformed/malicious HTTP
- SSL/TLS termination

Not included in sample Flask project

- Try it yourself!



Gunicorn + Nginx

Use both gunicorn and Nginx, for great justice!

Nginx serves static

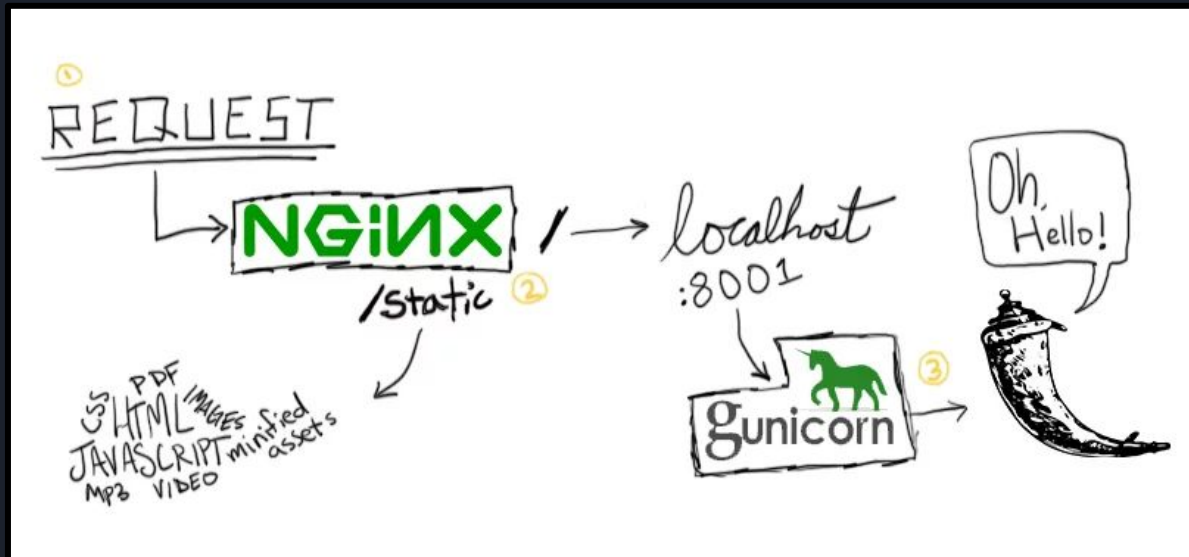
gunicorn implements WSGI

Flask handles app logic



Do it and your friends **WILL** think you're cool

Gunicorn + Nginx



Hosting using Docker



Configuring many separate services can be difficult to deploy. Containers can make setup and deployment easier.

<https://github.com/tiangolo/uwsgi-nginx-flask-docker>

Next Steps - Cloud Deployment

Are you interested in learning about Cloud technology? Now's a great time to try it and learn! You can deploy Flask applications with AWS Elastic Beanstalk or Azure App Services.

Free when you use student credits! (GitHub Education Pack)

You can also deploy to any virtual machine, including those hosted on AWS, Azure, Google Cloud Platform, DigitalOcean, etc. ...





Questions?
Comments?
Concerns?

Anything you would like me to explain in the example application?



References

The Flask documentation is great!

<http://flask.pocoo.org/docs/1.0/>