
University of Colorado Boulder

Topics

- Automating with Flask

Why

- As a network engineer you will want to automate simple tasks.
- You do not want to be the one that has to run these command all the time.
- Push these commands off to junior members of the team.

Flask

- Flask is a light weight web server framework for python.
- It is quick and easy to deploy a web app.
- Can call python scripts natively without help of system calls or cgi.
- More secure as it is a single process and not a cobbled together solution.

Flask install

- pip install flask

Install Flask

```
Collecting flask
/usr/lib/python2.7/site-packages/pip-8.1.2-py2.7.egg/pip/_vendor/requ
I (Subject Name Indication) extension to TLS is not available on this
ion failures. You can upgrade to a newer version of Python to solve t
ing.
  SNIMissingWarning
/usr/lib/python2.7/site-packages/pip-8.1.2-py2.7.egg/pip/_vendor/requ
lable. This prevents urllib3 from configuring SSL appropriately and m
 For more information, see https://urllib3.readthedocs.org/en/latest/
  InsecurePlatformWarning
  Downloading Flask-0.11.1-py2.py3-none-any.whl (80kB)
    100% |################################| 81kB 181kB/s
Collecting itsdangerous>=0.21 (from flask)
  Downloading itsdangerous-0.24.tar.gz (46kB)
    100% |################################| 51kB 207kB/s
Collecting Werkzeug>=0.7 (from flask)
  Downloading Werkzeug-0.11.11-py2.py3-none-any.whl (306kB)
    100% |################################| 307kB 84kB/s
Collecting Jinja2>=2.4 (from flask)
```

Flask Hello World

```
#!/usr/bin/env python
from flask import Flask

app = Flask(__name__)

@app.route('/')
def index():
        return("Hello World")

if __name__ == '__main__':
        app.debug = True
        app.run(host='0.0.0.0', port=80)
```

Flask

- Flask uses templates and static files for the bulk of your HTML responses.
- They are located in /static and /templates

Web Server Conventions

- The Document Root is the root of where your web server serves files from.
- The web server looks for files in /var/www/html .
- This makes your URLs
- /var/www/html/index.html would be accessed through the browser like /index.html .
- In Flask, it is wherever you execute the script.

Flask

```
from flask import Flask, render_template, Markup, request
import sys
```

Flask

- First we imported the Flask class. An instance of this class will be our WSGI application.
- Next we create an instance of this class.
- We then use the route() decorator to tell Flask what URL should trigger our function.
- The function is given a name which is also used to generate URLs for that particular function, and returns the message we want to display in the user's browser.

Flask

```
root@jmarduino2:~# python server.py
 * Running on http://0.0.0.0:80/ (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger pin code: 856-521-564
10.0.1.2 - - [23/Sep/2016 11:05:41] "GET / HTTP/1.1" 200 -
```

Creating Pages

- To create a page you create a route and define it.

```
@app.route('/secondPage')
def secondPage():
        return("You've hit the second page")
```

Handling Variables

```
@app.route('/third/<int:n>')
def third(n):
        return("You sent an int " + str(n))
```

- http://jmarduino2.local/third/3

Handling Variables

```
@app.route('/fourth/<string:s>')
def fourth(s):
        return("You sent an string " + s)
```

Handling Variables

```
@app.route('/fifth/<msg>')
def fifth(msg):
        return("You sent a bunch of things " + msg)
```

SSH

- **SSH is used on most network equipment and Linux/Unix hosts.**
- **We will use the paramiko tool kit to connect over SSH.**

SSH Host Keys

```
bash-3.2# ssh root@10.0.1.2
The authenticity of host '10.0.1.2 (10.0.1.2)' can't be established.
RSA key fingerprint is ea:05:57:de:1e:e5:ee:70:60:0e:0e:51:3d:88:23:97.
Are you sure you want to continue connecting (yes/no)?
```

Connecting

```
>>> import paramiko
>>> client = paramiko.SSHClient()
>>> client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
>>> client.connect('10.0.1.2', username='root', password='password')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/Library/Python/2.7/site-packages/paramiko/client.py", line 380, in connect
    look_for_keys, gss_auth, gss_kex, gss_deleg_creds, gss_host)
  File "/Library/Python/2.7/site-packages/paramiko/client.py", line 621, in _auth
    raise saved_exception
paramiko.ssh_exception.AuthenticationException: Authentication failed.
```

Executing

```
>>> ssh_session = client.get_transport().open_session()
>>> if ssh_session.active:
...     ssh_session.exec_command('df -h')
...     print ssh_session.recv(1024)
...
Filesystem       Size  Used Avail Use% Mounted on
/dev/root        1.4G  1.1G  288M  79% /
devtmpfs         480M     0  480M   0% /dev
tmpfs            481M     0  481M   0% /dev/shm
tmpfs            481M  572K  480M   1% /run
tmpfs            481M     0  481M   0% /sys/fs/cgroup
tmpfs            481M  4.0K  481M   1% /tmp
/dev/mmcblk0p10  1.3G   19M  1.3G   2% /home
tmpfs            481M  6.1M  474M   2% /var/volatile
/dev/mmcblk0p5  1003K   19K  913K   3% /factory
/dev/mmcblk1p1    29G   32K   29G   1% /media/sdcard
/dev/loop0       767M  908K  766M   1% /media/storage
tmpfs             97M     0   97M   0% /run/user/0
```

SSH

- Why not make a web form that prompts for a username, password and IP to back up switch configs.
- Store the configs in a DB and analyze for changes.

Creating Pages

- Flask uses templates. Create a templates directory and a file in it.

```
<html>
 <head>
  <title> Intro to Flask </title>
 </head>
 <body>

{% if bodyText%}
  {{ bodyText }}
{% endif %}

 </body>
</html>
```

Using Templates

```
from flask import Flask, render_template, Markup


@app.route('/sixth')
def sixth():
        bodyText=Markup("<b> Hello HTML </b> ")
        return render_template('template.html', bodyText=bodyText)
```

Templates

```
1  <html>
2   <head>
3    <title> Intro to Flask </title>
4   </head>
5   <body>
6
7
8    <b> Hello HTML </b>
9
10
11  </body>
12 </html>
```

Better looking sites

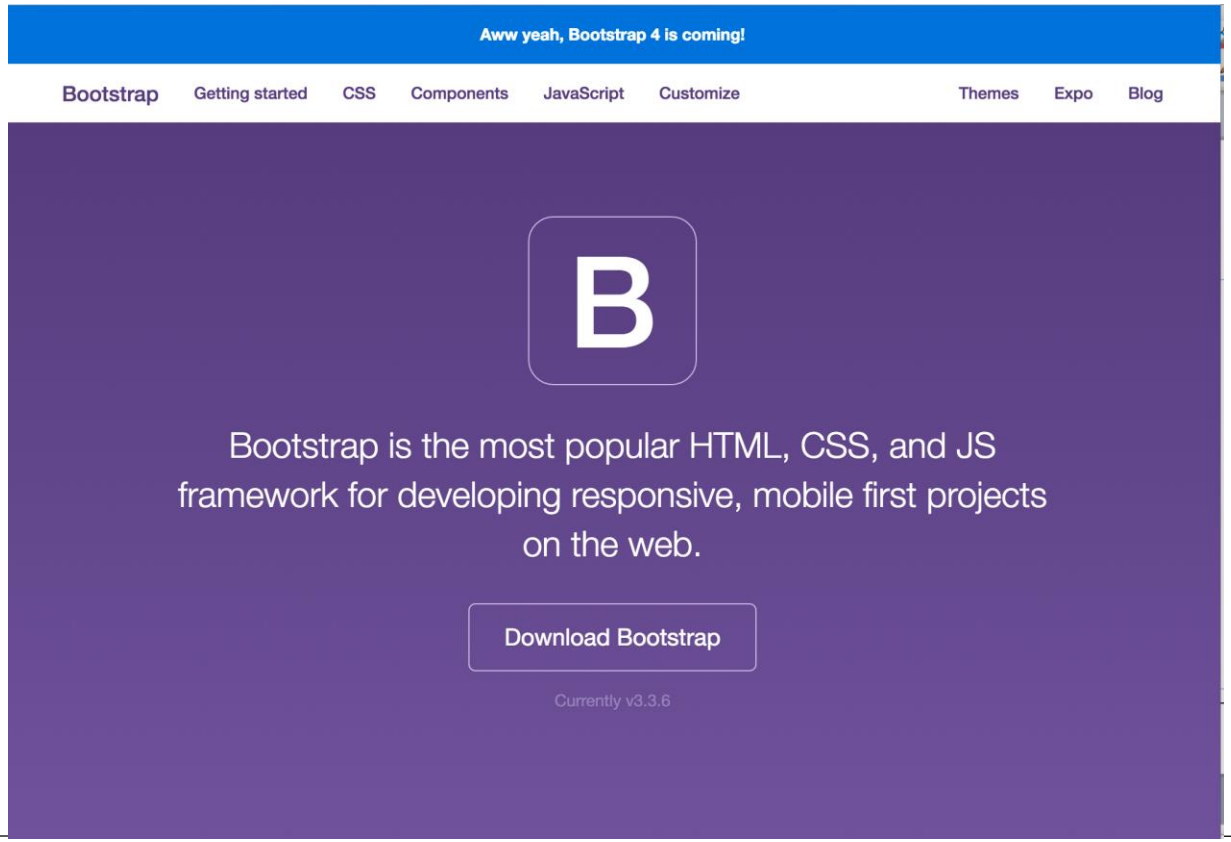
- I am not a graphic designer.
- But neither are a lot of people.

Rather let me fail then never to have tried at all. Rule every moment, seize every day.

- Twitter Bootstrap

Bootstrap

- http://getbootstrap.com/



Bootstrap

- Bootstrap is a framework for you to deploy apps.
- Copy source to your machine, put in DocumentRoot.
- Begin by playing with examples:
- http://getbootstrap.com/getting-started/#examples

Getting Started

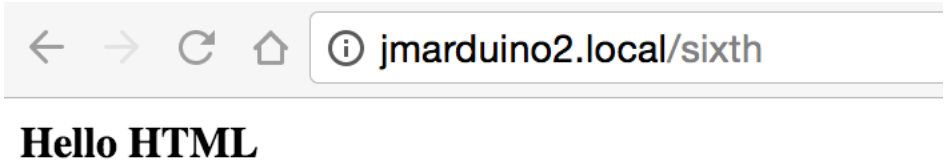
- Do a HelloWorld with http://getbootstrap.com/examples/starter-template/

Theme

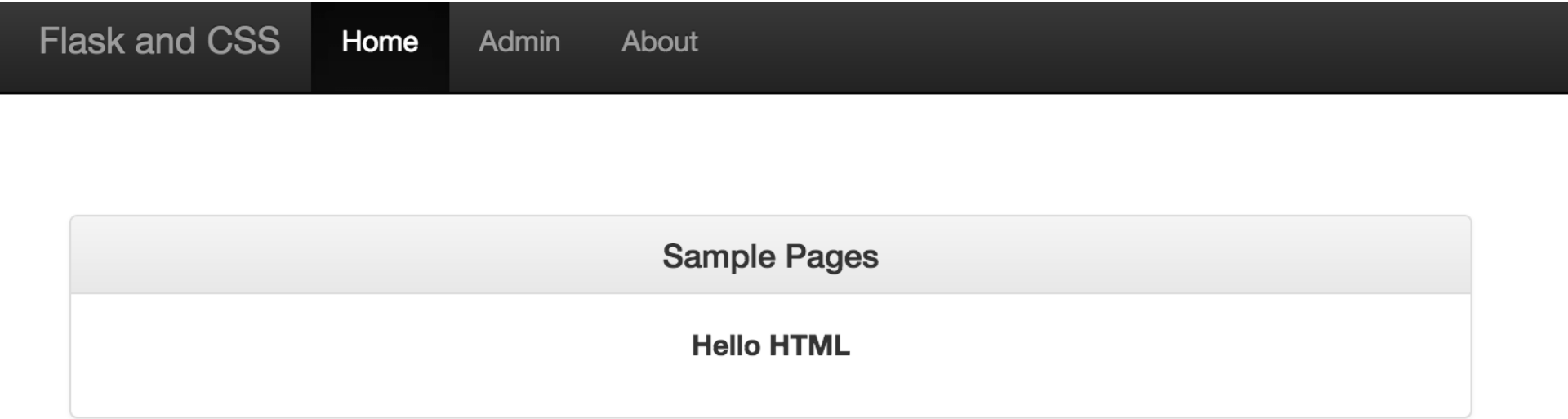
- More options on the "theme" http://getbootstrap.com/examples/theme/

- Add tables, buttons, progress bars.

Old Look



New Look



Simple

- Just move the {{bodyText}} in to the center of the template.

```
<div class="container">
  <div class="starter-template">
   <p class="lead">
      <div class="panel panel-default">
        <div class="panel-heading">
          <h3 class="panel-title"> Sample Pages </h3>
        </div>
        <div class="panel-body">
          {% if bodyText%}
                    {{ bodyText }}
        {% endif %}
        </p>
  </div>
```

Using SSL

- HTTP transmits data in clear text.
- HTTPS uses SSL to encrypt data in transit.

- Two methods.
  - adhoc - Creates it's own certs.
  - context - You create a cert (or buy)

Adhoc

- opkg update
- opkg install pyopenssl
- vi server.py
- remove `app.run(host='0.0.0.0', port=80)`

```
app.run(host='0.0.0.0', debug=False, port=443, ssl_context='adhoc')
```

OpenSSL

- opkg install openssl-util
- openssl genrsa -des3 -out server.key 1024
- openssl req -new -key server.key -out server.csr
- openssl rsa -in server.key -out server.key

Run as a service

- To run as a service we will us gunicorn
- pip install gunicorn

- gunicorn --bind 0.0.0.0:443 server:app -p /var/run/flask.pid --daemon

Start/stop

Create a startup script, place in /etc/init.d

```
#!/bin/sh

case $1 in

start)
        gunicorn --bind 0.0.0.0:80 server:app -p /var/run/flask.pid --daemon
;;
stop)
        kill `cat /var/run/flask.pid`
;;
*)
        echo "Usage: $0 stop|start"
;;
esac
```

Flask variables

- We have provided all of the variables at this point. Lets collect variables from the user.

Get/Post

- To send data to a server you will use a HTML form.

<from method=post action=/myPage>

<input type=text name=myVar> </input

<input type=submit name=submit value=submit>

</form>

Get/Post

```
@app.route('/myPage', methods=['GET', 'POST'])
def myPage():
        myVar=request.form['myVar'])
```

logging

- Import the logging module
- Start the logger and define where to log.

```
logging.basicConfig(filename='/var/log/flask.log',level=logging.DEBUG)
logging.debug("Started App")
```

error handling pages

- Use @app.errorhandler(errorCode)
- Popular codes: 404, 401,500

```
@app.errorhandler(404)
@app.errorhandler(500)
def errorpage(e):
        return("Oops, something went wrong")
```

Sessions

- Sessions are used to pass variables from one request to the next.
- This is implemented on top of cookies for you and signs the cookies cryptographically.
- What this means is that the user could look at the contents of your cookie but not modify it, unless they know the secret key used for signing.

Sessions require a key

- Sessions require a secret key.
- Generate one:
  - date | shasum -a 256 | cut -d" " -f1
- app.secret_key="theAboveOutput"

Sessions

- Session variables are a key value pair.
- i.e. session['foo']=bar
- Check if they exist before accessing
- if sessions.get('foo')
- if session['foo'] = "bar":

Authentication

- To authenticate we will check a username and password against a SQLite DB.
- We will store a hash of the password as an introduction.
- It is not good enough though. We will go on to store a per user salt and authenticate that way.

Authentication

- Creating a DB in sqlite
- sqlite3 dbName.db
- create table …

Lets talk about passwords

- Do we store a password in plain text?
- Do we store a hash?
- Do we store a hash and salt?
- Do we use a site wide salt?

Passwords

- What we should do it create a per user salt.
- Add the salt to the password and hash that.
- Use sha512.
- This way if the password file gets compromised they would have to brute force each account creating a new rainbow table for each salt.

User Table

- sqlite> .schema
- CREATE TABLE users(username varchar(128), password varchar(512), salt varchar(512));

Logging in

- When the user logs in, you will select a username, hashed password and salt from the database.
- The password from the user has the salt added and then hashed.
- If the stored password and newly created hash match. Create a session and login.

Logging in

- But first lets make an easier login.
- CREATE TABLE users(id int, username varchar(128), password varchar(128));

- We will store a sha512 of the password.
- We'll use salts later.
- Select from the DB the user where the hash of the password equals the stored hash.

Creating the hash

- Python's hashlib does not match *nix command line tools.

```
sazed:flask joe$ echo "flaskRules"|shasum -a 512
eff8795999df0cc5b0ff4babbf3866129a6d491b90ab603c5459945634eb75f52d9e72742c122d81
24ca7051ca43363215ae5c12721c22e3c3b5c5ceea067c05  -
```

Hashed password using Sha 512 :

7280b90eebd22280ba87d088d08da4e0325a075c1722acf071ff8a5392218bde8d7941eeb9ab289c2862d25e89677af633f4c7dc3a0067d6fb641110a4711ff3

- To create your first account create a page to show the hash.

Creating the hash

- To create your first account create a page to show the hash.
- Create a hash function.

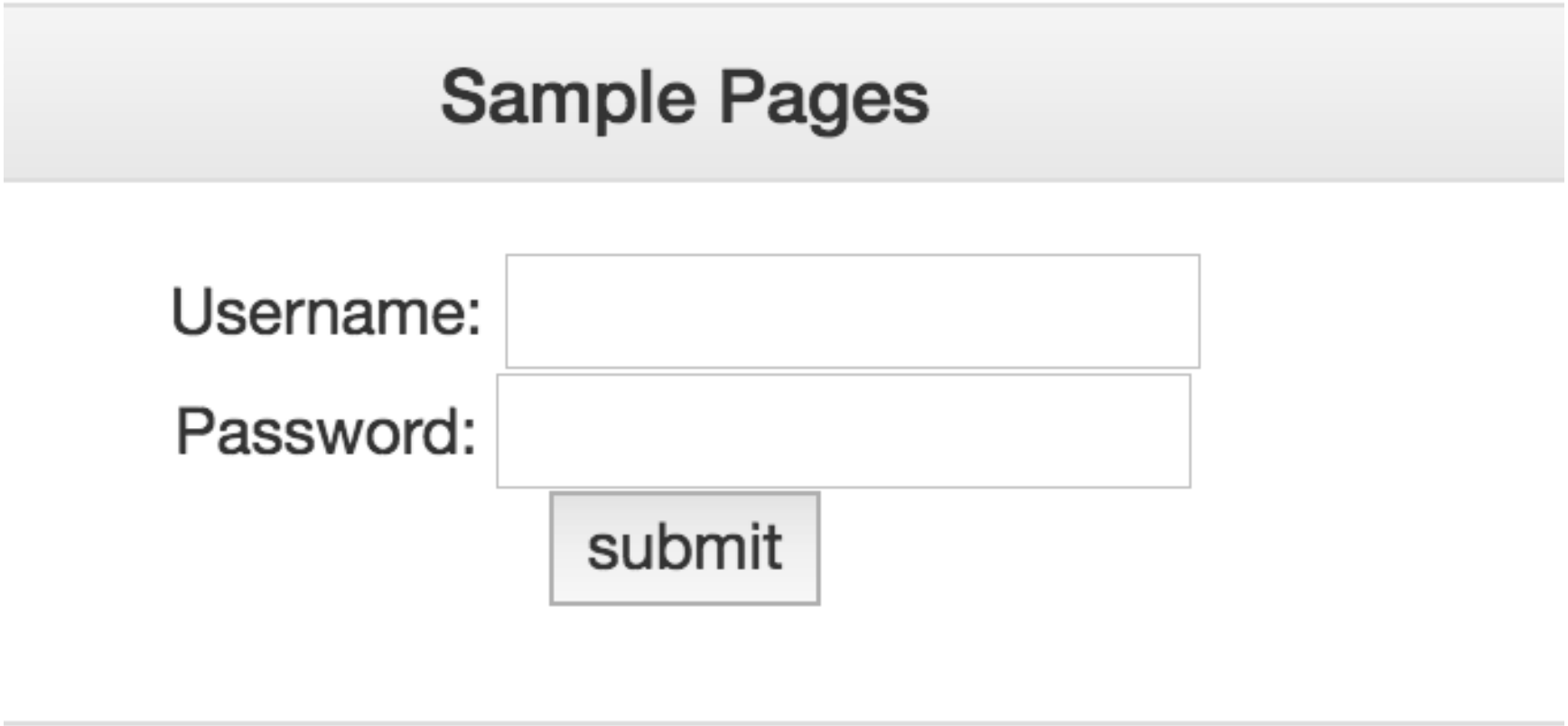
```
def getHash(passText):
        hashPass=hashlib.sha512()
        hashPass.update(passText)
        return(hashPass.hexdigest())
```

Adding First Account

- Add the account to SQLite

```
sqlite> insert into users (id, username, password) values('', 'joe', '7280b90eeb
d22280ba87d088d08da4e0325a075c1722acf071ff8a5392218bde8d7941eeb9ab289c2862d25e89
677af633f4c7dc3a0067d6fb641110a4711ff3');
```

Create a login page



Query db

```
@app.route('/login', methods=['GET', 'POST'])
def login():
        db = sqlite3.connect('server.sql3')
        db.row_factory = sqlite3.Row
        epass=getHash(request.form['postPass'])
        query="select username, password from users where username=? and password=?"
        t=(request.form['postUser'], epass)
        cursor=db.cursor()
        cursor.execute(query, t)
        rows = cursor.fetchall()
        if len(rows) == 1:
                bodyText=request.form['postUser'] + " " + request.form['postPass']
                bodyText=bodyText + " Success!"
                session['authenticated']='yes'
        else:
                bodyText = "Incorect Login."
```

Check Session

```
@app.route('/admin/<int:i>')
def admin(i):
        if session.get('authenticated'):
                if session['authenticated'] != 'yes':
                        response=redirect('/loginForm', code=302)
                        return response
        else:
                response=redirect('/loginForm', code=302)
                return response

        bodyText=("You have verified that you are authenticated")
        return render_template('templatecss.html', bodyText=bodyText)
```

Logout

```
@app.route('/logout')
def logout():
        session['authenticated']='no'
        response=redirect('/', code=302)
        return response
```

Improvements

- This so far is just an intro.
- We should add a groups table.
- Sessions should be improved:
  - Add a timeout
  - Add an origin check
  - Add a USER_AGENT check
  - Add a client IP check