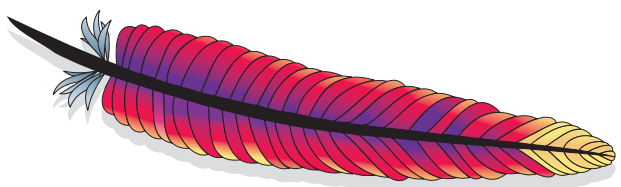
Django + httpd

Jeff Trawick

http://emptyhammock.com/

May 12, 2016

ApacheCon NA 2016



Get these slides...

`http://emptyhammock.com/projects/info/slides.html`

Get a fresh copy of the slide deck before using any recipes. If I find errors before this deck is marked as superseded on the web page, I'll update the .pdf and note important changes here. (And please e-mail me with any problems you see.)

Who am I?

Essentially:

Sometimes a web server guy, sometimes a web application guy

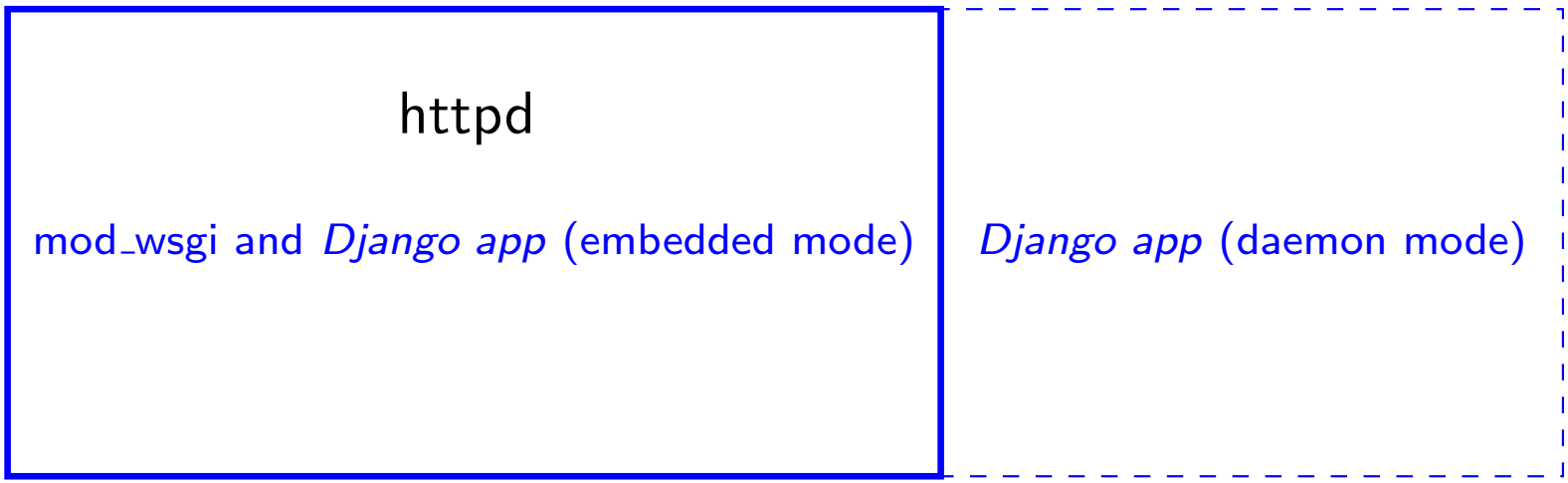
Common ways to deploy Python applications

- httpd + mod_wsgi + *Django app*
- httpd + mod_proxy/mod_proxy_*protocol* + (uWSGI or Gunicorn) + *Django app*
- nginx + *proxy protocol* + (uWSGI or Gunicorn) + *Django app*

The nginx flavor is essentially the same as the second httpd flavor.

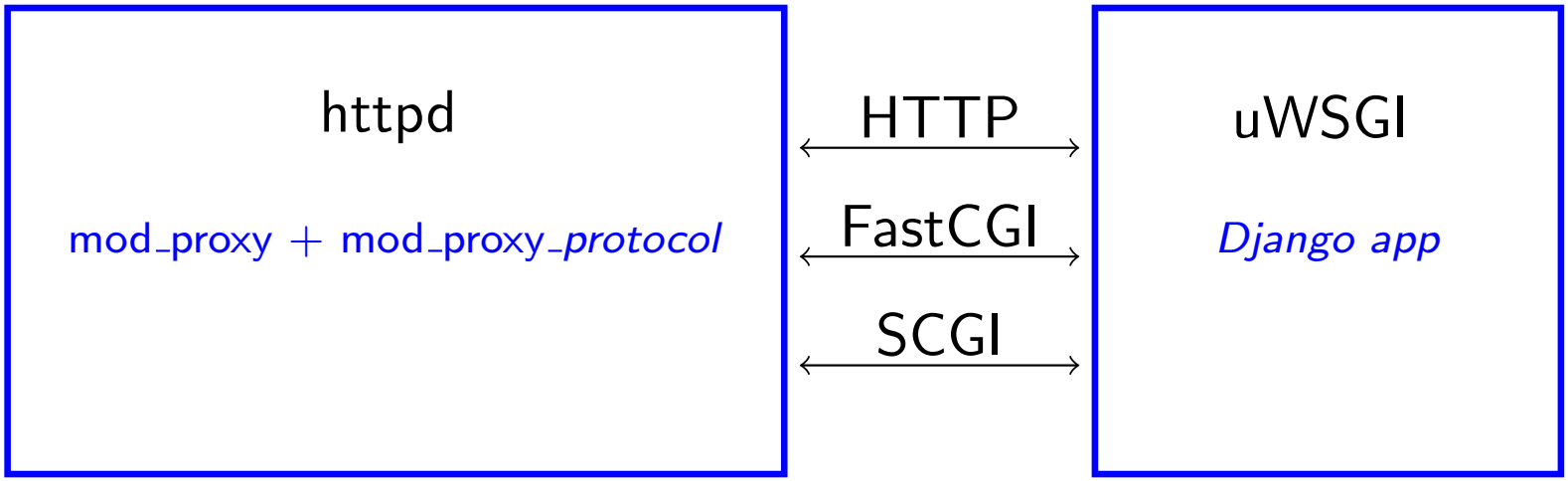
Django app is Python + Django + libraries + your application.

mod_wsgi



- *Django app* can run inside or outside of httpd processes (*embedded* or *daemon*)
- No concerns about lifecycle of *Django app* since it matches that of httpd — great for small scripts that don't warrant much effort

mod_proxy + mod_proxy_*protocol*



- uWSGI/Gunicorn largely interchangable
- httpd/nginx largely interchangable
- Choice of wire protocols between web server and application
- Lifecycle of uWSGI has to be managed in addition to that of web server

mod_wsgi vs. mod_proxy-based solution

Reasons you may want to move beyond mod_wsgi:

- mod_proxy supports more separation between web server and application.
  - Moving applications around or running applications in different modes for debugging without changing web server
  - Changes to the web front-end without affecting application
  - No collision between software stack in web server vs. software stack in application (e.g., different OpenSSL versions)
- mod_proxy has a lot of shared code, configuration, and concepts that are applicable to other application hosting.
- mod_wsgi doesn't support WebSockets.

Choices within the mod_proxy space

Further choices arise once mod_proxy is selected:

- Wire protocol (HTTP, FastCGI, or SCGI)
- Socket transport (TCP or Unix)
- Load balancing
- Application container (uWSGI, Gunicorn, etc.)

HTTP vs. FastCGI vs. SCGI

- Speed (with httpd)
  - SCGI faster than FastCGI
  - FastCGI faster than HTTP
- Speed (with nginx) SCGI, FastCGI, HTTP pretty close (significantly lower requests/sec than httpd with FastCGI or SCGI for the workloads I tried)
- SCGI is by far the simplest protocol, and HTTP is by far the most complex.
- Encryption
  - HTTP supports encryption between web server and application, but the others do not.
- Tool support (telnet-as-client, Wireshark, etc.)

What SCGI looks like

```
int main(int argc, char **argv)
{
  socket(); bind(); listen();
  while (1) {
    int cl = accept();
    read(cl, buf);
0x0000: 3233 3433 3a43 4f4e 5445 4e54 5f4c 454e  2343:CONTENT_LEN
0x0010: 4754 4800 3000 5343 4749 0031 0055 4e49  GTH.0.SCGI.1.UNI
0x0020: 5155 455f 4944 0056 5764 4f50 5838 4141  QUE_ID.VWdOPX8AA
0x0030: 5145 4141 476d 5745 6751 4141 4141 4c00  QEAAGmWEgQAAAAL.
0x0040: 7072 6f78 792d 7363 6769 2d70 6174 6869  proxy-scgi-pathi
...
    write(cl, buf);
0x0000: 5374 6174 7573 3a20 3230 3020 4f4b 0d0a  Status: 200 OK..
0x0010: 582d 4672 616d 652d 4f70 7469 6f6e 733a  X-Frame-Options:
0x0020: 2053 414d 454f 5249 4749 4e0d 0a43 6f6e   SAMEORIGIN..Con
0x0030: 7465 6e74 2d54 7970 653a 2074 6578 742f  tent-Type: text/
0x0040: 6874 6d6c 3b20 6368 6172 7365 743d 7574  html; charset=ut
...
    close(cl);
  }
}
```

sysdig hint

```
sudo sysdig -X proc.name=httpd and fd.num=37
```

TCP sockets vs. Unix sockets

- With both httpd and nginx, for all protocols tested, Unix sockets[1] are noticeably faster than TCP.
- The more complex Unix socket permissions can be a blessing or a curse.
- TCP supports distribution among different hosts.
- TCP consumes kernel resources (and confuses many users of netstat) while sockets remain in TIME_WAIT state.
- TCP's requirement for *lingering close* can require more server (application container) resources.

[1] Unix socket support in mod_proxy for HTTP and FastCGI requires httpd 2.4.7 or later; Unix socket support in mod_proxy for SCGI requires httpd 2.4.10 or later.

Some cases with simple decision-making

- If **speed** is of absolute concern, pick **SCGI** with **Unix** sockets.
- If **interoperability** of your application stack for diagnostics or any other purpose is of absolute concern, pick **HTTP** with **TCP** sockets.
- If **encryption** between the web server and application is of absolute concern, pick **HTTP**.
- If **securing** your application stack from other software in your infrastructure is of absolute concern, and your application and web server run on the same host, pick **anything with Unix sockets**.

For this talk

SCGI with TCP sockets between httpd and the application[2]

```
LoadModule proxy_module modules/mod_proxy.so
LoadModule proxy_scgi_module modules/mod_proxy_scgi.so
```

[2] Mostly... Our WebSocket example is different.

SCGI differences between httpd 2.2 and 2.4

mod_proxy_scgi in 2.4

- Requires `proxy-scgi-pathinfo` envvar to be set in order to set `PATH_INFO` as required for many Python applications
- Adds support for Unix sockets (2.4.10)
- Supports internal redirects via arbitrary response headers set by the application (2.4.13)
- Passing authentication headers to the application *sanely*
- Any generic features added to mod_proxy in 2.4

Differences between 2.4.*something* and 2.4.*current*

I.e., mod_proxy_scgi improvements after some level of Ubuntu, for example

Ubuntu 14.04 has 2.4.7; Ubuntu 16.04 has 2.4.18; *current upstream* is 2.4.20

Caught in the middle?

- 2.4.10 adds support for Unix sockets
- 2.4.13 supports internal redirects via arbitrary response headers set by the application
- 2.4.13 adds `CGIPassAuth`

See `https://wiki.apache.org/httpd/Get24` for hints on which distros bundle which levels of httpd.

PPA for httpd 2.4.latest

`https://launchpad.net/~ondrej/+archive/ubuntu/apache2`

- `ppa:ondrej/apache2`
- From Ondřej Surý
- Tracks httpd 2.4.x
- Currently has 2.4.20 for Ubuntu *precise*, *trusty*, *wily*, and *xenial*

Minimal build of httpd 2.4 to support Python applications

If you need to build from source

```
./configure \
--with-included-apr      --enable-nonportable-atomics \
--enable-exception-hook \
--enable-mpms-shared=all --enable-mods-shared=few \
--enable-expires=shared  --enable-negotiation=shared \
--enable-rewrite=shared  --enable-socache-shmcb=shared \
--enable-ssl=shared      --enable-deflate=shared \
--enable-proxy=shared    --enable-proxy-scgi=shared \
--disable-proxy-connect  --disable-proxy-ftp \
--disable-proxy-http     --disable-proxy-fcgi \
--disable-proxy-wstunnel --disable-proxy-ajp \
--disable-proxy-express  --disable-lbmethod-bybusyness \
--disable-lbmethod-bytraffic \
--disable-lbmethod-heartbeat
```

(But keep `proxy-http` and `proxy-wstunnel` for WebSockets.)

Building blocks on the application side

- Django for the web application framework
- uWSGI for the "container" that hosts/manages the application processes, along with
  - An init script to start/stop the application by controlling uWSGI
  - A uWSGI configuration file

Where is the sample code?

`https://github.com/trawick/httpd.py`

(branch `AC2016`)

You'll see snippets on later slides.

Simplest little bit of Django

```
from django.http import HttpResponse

PATH_VARS = ('PATH_INFO', 'PATH_TRANSLATED', 'SCRIPT_FILENAME',
             'REQUEST_URI', 'SCRIPT_URI')

def cgivars(request):
    return HttpResponse('<br />'.join(map(lambda x: '%s => %s' %
        (x, request.environ.get(x, '&lt;unset&gt;')), PATH_VARS))
    )

urlpatterns = [
    url(r'^cgivars/$', views.cgivars),
]
```

```
Listen 18083
<VirtualHost 127.0.0.1:18083>
    # Lots of stuff inherited from global scope
    SetEnvIf Request_URI . proxy-scgi-pathinfo
    ProxyPass /app/ scgi://127.0.0.1:3006/
</VirtualHost>
```

Running the Django app via uWSGI

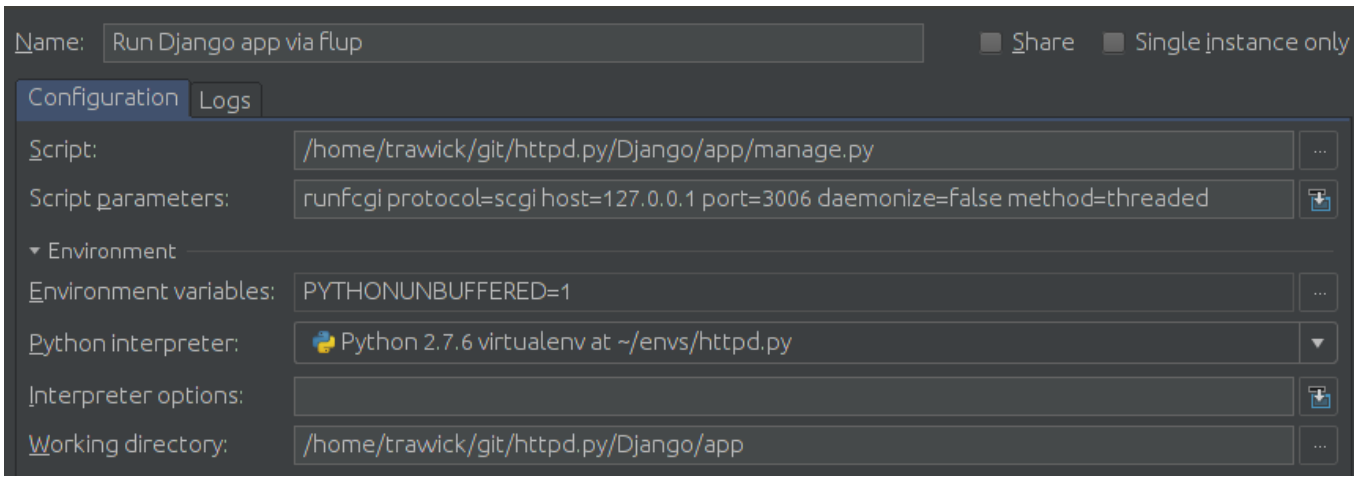
simple script for running in the foreground

- *terminal or PyCharm^H^H^H^H^H^H^HIDE, but not deployment*

```
VENV=/home/trawick/envs/httpd.py
${VENV}/bin/uwsgi --scgi-socket 127.0.0.1:3006 \
  --module app.wsgi \
  --chdir /home/trawick/git/httpd.py/Django/app \
  --virtualenv ${VENV}
```

Running the Django app in simple Python container

Sometimes you need to debug your app in a deployment-like scenario, such as with a web server front-end.



- flup is pure Python, so you can attach for debugging in the usual manner
- Uses the same protocol as production deployment
- May need to tweak processes/threads to make it easy to debug a request

X-Sendfile to offload file serving to the web server

```
from django.http import HttpResponse

def sendfile(request):
    filename = request.environ['DOCUMENT_ROOT'] + '/' + 'bigfile.html'
    response = HttpResponse()
    response['X-Sendfile'] = filename
    return response

urlpatterns = [
    url(r'^sendfile/$', views.sendfile),
]

# add to .conf for httpd:
ProxySCGISendfile On
```

X-Location to offload request after application authorizes it

```
def protected(request):
    filename = '/static/protected/index.html'
    response = HttpResponse()
    # Django prior to 1.9 will turn this
    # into Location: http://127.0.0.1:18083/static/protected/foo
    #     response['Location'] = filename

    # This is passed through unadulterated:
    response['X-Location'] = filename
    return response

# add to .conf for httpd:
ProxySCGIInternalRedirect X-Location
ProxyPass /static/protected/ !
...
# Only allow access to /static/protected/ if a request to /app/protected/
# redirected there.  (I.e., must have been redirected, must have hit
# the app first)
<Location /static/protected/>
    Require expr %{reqenv:REDIRECT_REQUEST_URI} =~ m#^/app/protected/#
</Location>
```

Streaming a response through web server

```
from django.http import StreamingHttpResponse

def export_stacktraces(request):

    def generate_response(qs):
        yield '['
        need_comma = False
        for st in qs:
            if need_comma:
                yield ','
            yield json.dumps(st.raw)
            need_comma = True
        yield ']'

    resp = StreamingHttpResponse(
        generate_response(Stacktrace.objects.filter(owner=request.user)),
        content_type='application/json'
    )
    resp['Content-Disposition'] = 'attachment; filename=stacktraces.json'
    return resp
```

Handling /static/ for apps

With the proper preparation, Django's `./manage.py collectstatic` will collect static files into a location that the web server knows about and can serve.

```
Alias /static/ {{ static_dir }}/
...
ProxyPass /static/ !
...
<Directory {{ static_dir }}/>
    Require all granted
    # only compress static+public files (see BREACH)
    SetOutputFilter DEFLATE
    # if they aren't naturally compressed
    SetEnvIfNoCase Request_URI \.(?:gif|jpe?g|png)$ no-gzip
    ExpiresActive On
    ExpiresDefault "access plus 3 days"
    Header set Cache-Control public
</Directory>
```

Consider something similar for `/media/`.

`robots.txt` in `/static/` too?

```
Alias /robots.txt {{ static_dir }}/robots.txt
...
ProxyPass /robots.txt !
...
```

Consider something similar for `/favicon.ico`.

Add load balancing

```
LoadModule proxy_balancer_module modules/mod_proxy_balancer.so
LoadModule lbmethod_byrequests_module modules/mod_lbmethod_byrequests.so

ProxyPass /app/ balancer://app-pool/
<Proxy balancer://app-pool/>
  BalancerMember scgi://127.0.0.1:10080
  BalancerMember scgi://127.0.0.1:10081
  # The server below is on hot standby
  BalancerMember scgi://127.0.0.1:10082 status=+H
  ProxySet lbmethod=byrequests
</Proxy>
```

(Also has a “balancer manager” which can be used to change settings dynamically)

I/O timeouts

- By default, the I/O timeout is the value of the `Timeout` directive (i.e., same as client I/O timeout).
- `ProxyTimeout` overrides that for proxy connections.
- Max time without being able to read one byte when trying to read (or similar for write)
  - This covers application time to build the response.

Recovery from backend errors

`ProxyPass` options:

- `retry=`*`seconds`* specifies the time before sending another connection to a previously-unhealthy application (e.g., ECONNREFUSED)
  - No other load balanced instances? You probably want this much lower than the default, 60 seconds.
- For balancer members: `failonstatus=`*`nnn,nnn,...`* will also treat the specified HTTP status codes from the application as indicating that it is unhealthy

Handling Basic auth in the application

- Apps commonly use form+cookie-based auth.
- Basic auth handled by the application may be useful.
- Normally httpd hides `Authorization` and `Proxy-Authorization` request headers from applications (can be subverted).
- mod_wsgi provides the `WSGIPassAuthorization` directive to enable that.
- `CGIPassAuth`[3] directive enables this cleanly for all CGI-like interfaces.

```
<Location /legacy-reports/>
    CGIPassAuth On
</Location>
```

[3] httpd $\geq$ 2.4.13

WebSockets

- Long-running, lightweight connections
  - Little if any overhead imposed on performance-sensitive application (e.g., games)
  - Little if any overhead imposed on infrastructure to maintain lots of these connections
- Kept alive by browser and application container (ping and pong)
- Application code in browser and application only wake up when necessary
- Set up when a special HTTP request is *upgraded* to a WebSocket tunnel between client and application
- HTTP proxies usually support WebSockets
- Requires HTTP from client to application, so no FastCGI or SCGI transport for the WebSocket data

Some Python WebSockets caveats

Now:

- WSGI doesn't encompass WebSockets
- No other finalized PEP/standard covers Python interface to WebSockets
- Not abundantly clear that current Django has a particular right way to do it
- Lack of interchangability of components in some cases (e.g., Flask-SocketIO works with Gunicorn but not with uWSGI)

Start looking at:

- WSGI-NG (`https://github.com/python-web-sig/wsgi-ng`)
- Django Channels (`https://channels.readthedocs.io/en/latest/`)

WebSockets example

- Uses HTTP + WebSockets extension between web server and application, instead of SCGI like in our other examples
- Uses uWSGI Python API instead of a container-agnostic API or framework like Django; this works around some of the caveats listed earier
  - web server configuration would be the same anyway

WebSockets example using uWSGI API

```
import uwsgi  # not installed in venv but works under uWSGI :(

html_template = <<<the JavaScript WebSocket client>>>

def application(env, sr):
    if env['PATH_INFO'] == '/':
        ws_scheme = 'ws'
        if 'HTTPS' in env or env['wsgi.url_scheme'] == 'https':
            ws_scheme = 'wss'
        sr('200 OK', [('Content-Type', 'text/html')])
        host = env.get('HTTP_X_FORWARDED_HOST', env['HTTP_HOST'])
        return index_html_template % (ws_scheme, host)
    elif env['PATH_INFO'] == '/ws/':
        uwsgi.websocket_handshake(env['HTTP_SEC_WEBSOCKET_KEY'],
                                  env.get('HTTP_ORIGIN', ''))
        while True:
            msg = uwsgi.websocket_recv()
            uwsgi.websocket_send(msg)
    else:
        sr('404 NOT FOUND', [('Content-Type', 'text/plain')])
        return 'Not found'
```

Some of the JavaScript code

See the template aspect of the JS snippet, as well as the I/O.

```
function init() {
  var s = new WebSocket("%s://%s/ws/");
  ...
  s.onopen = function() { s.send(i); }
  s.onmessage = function(e) {
    window.setTimeout(function () {
      s.send(i);
    }, 1500);
  }
  s.onerror = function(e) { ... }
  s.onclose = function() { ... }
}
window.onload = init;
```

Running the WebSocket app

simple script for running in the foreground

- *terminal or IDE, but not deployment*

```
VENV=/home/trawick/envs/httpd.py
# gevent parameter needed to support more than one WebSocket
# request (i.e., set up gevent)
${VENV}/bin/uwsgi --http-socket 127.0.0.1:3007 \
  --http-raw-body \
  --gevent 100 \
  --wsgi-file app.py \
  --chdir /home/trawick/git/httpd.py/uWSGI-websocket
```

.conf for proxying to the WebSocket app

```
Listen 18085

<VirtualHost 127.0.0.1:18085>
    # Lots of stuff inherited from global scope

    CustomLog logs/websocket-app-access.log common
    ErrorLog logs/websocket-app-error.log
    LogLevel warn

    # Note that /ws/ is the exception among all requests.
    # Put that first so that it won't be handled by HTTP.
    ProxyPass        /ws/ ws://127.0.0.1:3007/ws/
    ProxyPass        /    http://127.0.0.1:3007/
    ProxyPassReverse /    http://127.0.0.1:3007/
</VirtualHost>
```

End of WebSockets example

A more complete example .conf

for a non-WebSockets application, in the form of a Jinja2 template

```
<VirtualHost *:80>
    ServerName {{ canonical_server_name }}
    Redirect permanent / https://{{ canonical_server_name }}/
</VirtualHost>

<VirtualHost *:443>
    ServerName {{ canonical_server_name }}
    ServerAdmin me@example.com

    CustomLog {{ log_dir }}/httpd-access.log common
    ErrorLog {{ log_dir }}/httpd-error.log
    LogLevel {{ httpd_log_level }}
```

continued

A more complete example .conf

```
# DocumentRoot unused since / is proxied; point it
# to something users can access anyway
DocumentRoot {{ static_dir }}/

<Directory />
    Options FollowSymLinks
    Require all denied
    AllowOverride None
</Directory>
```

continued

A more complete example .conf

```
SetEnvIf Request_URI . proxy-scgi-pathinfo
ProxyTimeout 30
# ProxySCGISendfile On
# ProxySCGIInternalRedirect X-Location

Alias /robots.txt {{ static_dir }}/robots.txt
Alias /static/ {{ static_dir }}/
# Alias /media/ XXXXX

ProxyPass /robots.txt !
ProxyPass /static/ !
# ProxyPass /media/ !
ProxyPass / scgi://127.0.0.1:{{ application_port }}/ retry=5
```

continued

A more complete example .conf

```
<Directory {{ static_dir }}>
    Require all granted
    # only compress static+public files (see BREACH)
    SetOutputFilter DEFLATE
    # if they aren't naturally compressed
    SetEnvIfNoCase Request_URI \.(?:gif|jpe?g|png)$ no-gzip
    ExpiresActive On
    ExpiresDefault "access plus 3 days"
    Header set Cache-Control public
</Directory>
```

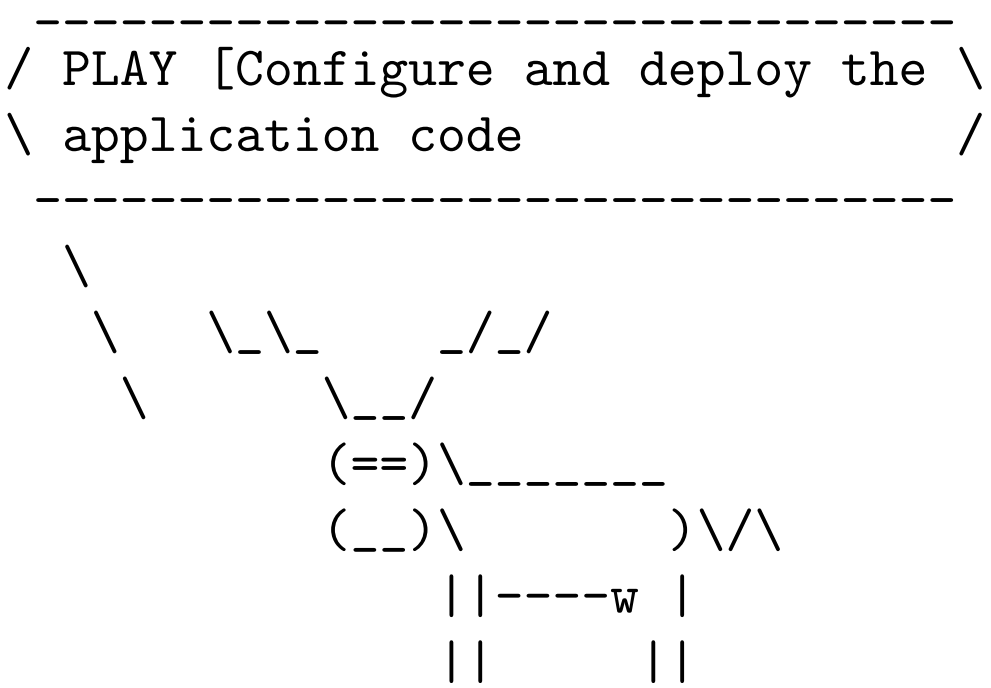
continued

A more complete example .conf

```
    SSLEngine on
    # SSL protocols/ciphers/etc. inherited from global scope

    Header always set Strict-Transport-Security "max-age=31536000"

    SSLCertificateKeyFile   /path/to/server.key
    SSLCertificateFile      /path/to/server.crt
</VirtualHost>
```


PLAY [Configure and deploy the
application code

Vagrant and Ansible

- Ansible for system configuration and application deployment
  - Same automation for staging, production, other images
  - Same automation whether system is provisioned with Vagrant or other tools
- Vagrant to automate creation of server VM
  - Automating mint machine together with configuration and deployment ensures that all aspects are covered.

`https://github.com/trawick/httpd.py/tree/AC2016/Django/deploy`

Features of the automation

The deployment consists of one Ubuntu server, the *webserver*, which runs the web server, Django application, and database.

- Create a user to own application resources, add to `sudoers`
- Install necessary system packages, as well as httpd-*latest* from a PPA
- Set up PostgreSQL user and database
- Create Python virtual environment with necessary libraries
- Configure httpd to route to application
- Configure uWSGI and its lifecycle to host application

Parts of the automation

`Vagrantfile`
: Create the machine, invoke Ansible

Ansible playbook `deploy.yml`
: Commands to configure system and deploy application

Ansible *hosts* file
: Variables specific to a particular server, such as passwords or IP addresses or . . .

Template files
: Various configuration files filled in with data specific to the deployment or server

Simplified file layout for example

```
./deploy.yml
./ansible/vagrant-hosts
./ansible/OTHER-hosts
./templates/init-script.j2
./templates/django-app.conf.j2
./templates/uwsgi-ini.j2
./Vagrantfile
```

(significantly simplified layout compared with many Ansible examples)

Invoking Vagrant and Ansible

- Bring up VM, create and/or provision as necessary

  ```
  $ vagrant up
  ```

- Re-provision existing VM

  ```
  $ vagrant provision
  ```

- Create new, provisioned VM from scratch, discarding one that already exists

  ```
  $ vagrant destroy -f ; vagrant up
  ```

- Invoke Ansible directly against a different host

  ```
  $ ansible-playbook -i ansible/OTHER-hosts deploy.yml
  ```

- See also `vagrant up`, `vagrant halt`, `vagrant suspend`, `vagrant ssh`, etc.

Using files directly from control host or from repo?

- Ansible config and templates/other files copied to server via Ansible come from git checkout on control host.
  - No need to push these changes to git repo before testing
- Application runs from git checkout on the server.
  - Must push application updates to git repo before re-deploying

Vagrantfile

```
Vagrant.configure(2) do |config|
  config.vm.box = "precise32"
  config.vm.box_url = "http://files.vagrantup.com/precise32.box"
  config.vm.network "private_network", ip: "10.10.10.15"
  config.vm.provision "ansible", run: "always" do |ansible|
    # ansible.verbose = "vvvv"
    ansible.limit = "all"
    ansible.playbook = "deploy.yml"
    ansible.inventory_path = "ansible/vagrant-hosts"
  end
end
```

- `precise32` is 32-bit Ubuntu 12 server
- Create entry in your `/etc/hosts` to map `simple-django.com` to 10.10.10.15

ansible/vagrant-hosts

```
[webservers]
vagrant ansible_ssh_host=127.0.0.1 ansible_ssh_port=2222

[webservers:vars]
initial_user=vagrant
log_dir=/tmp
pg_password=simple-django-db-password
git_repo_version=master
app_processes=1
app_threads=2
```

- 2222 is ssh port assigned by Vagrant for webserver VM

Overall structure of `deploy.yml`

```
---

- name: Create remote user
  hosts: webservers
  vars:
    remote_user: django-user
  remote_user: "{{ initial_user }}"
  sudo: true
  tasks:
    <<<create remote user, add to sudoers>>>

- name: Configure and deploy the application code
  hosts: webservers
  vars:
    remote_user: django-user
    application_port: 3006
    http_port: 80
    remote_checkout: /home/django-user/httpd.py
    <<<other settings>>>
  remote_user: "{{ remote_user }}"
  tasks:
    <<<remaining system and application configuration>>>
  handlers:
    <<<restart application and/or web server at end>>>
```

Install/update python-software-properties

```
- name: Make sure python-software-properties is installed
  apt: pkg=python-software-properties state=latest
  sudo: yes
```

```
 ------------------------------------------------------------
< TASK: Make sure python-software-properties is installed >
 ------------------------------------------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

```
changed: [vagrant]
```

Add PPA repo for httpd 2.4.*latest*

```
    - name: Add ppa repo for httpd 2.4.latest
      apt_repository: repo='ppa:ondrej/apache2/ubuntu'
      sudo: yes
```

```
 ----------------------------------------
< TASK: Add ppa repo for httpd 2.4.latest >
 ----------------------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

```
changed: [vagrant]
```

Install system packages

```
- name: Install packages
  apt: name={{ item }} state=latest
  sudo: yes
  with_items:
    - apache2
    - git
    - python-virtualenv
    - postgresql
    - libpq-dev
    - python-dev
    - python-psycopg2
```

```
 ------------------------
< TASK: Install packages >
 ------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

```
changed: [vagrant] => (item=apache2,git,python-virtualenv,postgresql,libpq-dev,...
```

Run httpd at boot

```
    - name: Make sure httpd is started and will run at boot
      service: name=apache2 state=started enabled=yes
```

```
 ------------------------------------------------------
< TASK: Make sure httpd is started and will run at boot >
 ------------------------------------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

```
ok: [vagrant]
```

Activate httpd modules

```
  - name: Configure system httpd to include various modules
    apache2_module: state=present name={{ item }}
    sudo: yes
    with_items:
      - proxy
      - proxy_scgi
      - headers
      - deflate
      - expires
    notify: restart system httpd
```

```
 ------------------------------------------------------
< TASK: Configure system httpd to include various modules >
 ------------------------------------------------------
       \   ^__^
        \  (oo)\_______
           (__)\       )\/\
               ||----w |
               ||     ||
```

```
changed: [vagrant] => (item=proxy)
changed: [vagrant] => (item=proxy_scgi)
changed: [vagrant] => (item=headers)
ok: [vagrant] => (item=deflate)
changed: [vagrant] => (item=expires)
```

Remove Debian/Ubuntu default vhost

```
    - name: Remove default virtualhost file.
      file:
        path: "/etc/apache2/sites-enabled/000-default.conf"
        state: absent
      sudo: yes
      notify: restart system httpd
```

```
 ----------------------------------------
< TASK: Remove default virtualhost file. >
 ----------------------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

```
changed: [vagrant]
```

Configure application vhost

```
    - name: Configure system httpd
      template: src=templates/django-app.conf.j2
                dest=/etc/apache2/sites-enabled/{{ project_name }}-vhost.conf
      sudo: yes
      notify: restart system httpd
```

```
 ------------------------------
< TASK: Configure system httpd >
 ------------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

```
changed: [vagrant]
```

Create uWSGI config directory

```
    - name: Create uWSGI config directory
      file: >
          dest={{ uwsgi_cfg_dir }}
          mode=755
          owner=root
          group=root
          state=directory
      sudo: yes
      notify: restart application
```

```
 ------------------------------------
< TASK: Create uWSGI config directory >
 ------------------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

```
changed: [vagrant]
```

Install/update uWSGI config

```
    - name: Add application uWSGI config
      template: src=templates/uwsgi-ini.j2
                dest={{ uwsgi_cfg_dir }}/{{ project_name }}.ini
      sudo: yes
      notify: restart application
```

```
 ------------------------------------
< TASK: Add application uWSGI config >
 ------------------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

```
changed: [vagrant]
```

Add application init script

```
    - name: Add application init script
      template: src=templates/init-script.j2
                dest=/etc/init.d/{{ project_name }}-app
                mode=0751
      sudo: yes
      notify: restart application
```

```
 ------------------------------------
< TASK: Add application init script >
 ------------------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

```
changed: [vagrant]
```

Configure run-levels for application

```
    - name: Configure run-levels for application
      command: update-rc.d {{ project_name }}-app defaults
      sudo: yes
```

```
 ------------------------------------------
< TASK: Configure run-levels for application >
 ------------------------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

```
changed: [vagrant]
```

Run application

```
    - name: Run application
      action: service name={{ project_name }}-app state=started
      sudo: yes
```

```
 ----------------------
< TASK: Run application >
 ----------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

```
changed: [vagrant]
```

Handler to restart application if needed

```
    - name: restart application
      service: name={{ project_name }}-app state=restarted
      sudo: yes
```

```
 ------------------------------
< NOTIFIED: restart application >
 ------------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

```
changed: [vagrant]
```

Handler to restart httpd if needed

```
    - name: restart system httpd
      service: name=apache2 state=restarted
      sudo: yes
```

```
 ---------------------------------
< NOTIFIED: restart system httpd >
 ---------------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

```
changed: [vagrant]
```

.conf template

```
{% if http_port != 80 %}
Listen {{ http_port }}
{% endif %}

<VirtualHost *:{{ http_port }}>
    ServerName simple-django.com
    # Lots of stuff inherited from global scope

    DocumentRoot {{ remote_checkout }}/Django/docroot
    <Directory {{ remote_checkout }}/Django/docroot/>
      Require all granted
    </Directory>

    CustomLog {{ log_dir }}/django-app-access.log common
    ErrorLog {{ log_dir }}/django-app-error.log
    LogLevel warn

    SetEnvIf Request_URI . proxy-scgi-pathinfo
    ProxySCGISendfile On
    <IfVersion >= 2.4.13>
    ProxySCGIInternalRedirect X-Location
    </IfVersion>

    ProxyPass /static/protected/ !
    ProxyPass /app/ scgi://127.0.0.1:{{ application_port }}/

    # Only allow access to /static/protected/ if a request to /app/protected/
    # redirected there.  (I.e., must have been redirected, must have hit
    # the app first)
    <Location /static/protected/>
        Require expr %{reqenv:REDIRECT_REQUEST_URI} =~ m#^/app/protected/#
    </Location>
</VirtualHost>
```

uWSGI configuration template

```
[uwsgi]
pidfile = {{ log_dir }}/{{ project_name }}.pid
daemonize = {{ log_dir }}/uwsgi-{{ project_name }}.log
scgi-socket = 127.0.0.1:{{ application_port }}
chdir = {{ django_src }}
module = app.wsgi
master = true
processes = {{ app_processes }}
threads = {{ app_threads }}
uid = {{ remote_user }}
gid = {{ remote_user }}
virtualenv = {{ virtualenv_dir }}
buffer-size = 8192
```

Init script template

```
#!/bin/sh

SERVICE_NAME={{ project_name }}-app
PIDFILE={{ log_dir }}/{{ project_name }}.pid
UWSGI_INI={{ uwsgi_cfg_dir }}/{{ project_name }}.ini
UWSGI_ENV={{ virtualenv_dir }}

. ${UWSGI_ENV}/bin/activate

<<<helper functions>>>

case "$1" in
  status)
    status_service
    ;;
  start)
    start_service
    ;;
  stop)
    stop_service
    ;;
  restart)
    if is_running; then
      stop_service
    fi
    start_service
    ;;
  *)
    echo "Usage: service $SERVICE_NAME {start|stop|restart|status}" >&2
    exit 1
    ;;
esac

exit 0
```

General httpd features which can be useful

- Web server cache (mod_cache, mod_disk_cache)
- Web server logging tricks
  - Configure httpd and application log formats to include `UNIQUE_ID`
  - Add response time (and maybe time to first byte[4]) in httpd access log
  - See `http://people.apache.org/~trawick/AC2014-Debug.pdf` for different tricks applicable to diagnosing application symptoms.
- Load balancing and mod_proxy balancer manager
- Monitoring capacity utilization for httpd and application

[4] mod_logio's `LogIOTrackTTFB` was added in 2.4.13.

Caktus Group project template

- Relatively complete application and infrastructure configuration
- Much more complex than the Ansible example, but handles many more requirements
- `https://github.com/caktus/django-project-template`
- Salt instead of Ansible
- nginx instead of httpd

A few things to add for the 5 hour version of this talk

- Django implementations of Basic auth
- Live load balancer demo, making dynamic changes via load balancer manager interface
- Current status of WSGI-NG and Django Channels, how to experiment with available code for Channels
- *Your ideas*

Thank you!