Deploying Python Applications with httpd

Jeff Trawick

http://emptyhammock.com/
trawick@emptyhammock.com

April 14, 2015

ApacheCon US 2015

Get these slides...

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

Revisions

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 If I find errors before this deck is marked as superseded on the web page, I'll update the .pdf and note important changes here.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?

- My day jobs over the last 25 years have included work on several products which were primarily based on or otherwise included Apache HTTP Server as well as lower-level networking products and web applications. My primary gig now is with a Durham, North Carolina company which specializes in Django application development.
- I've been an httpd committer since 2000. A general functional area of Apache HTTP Server that I have helped maintain over the years (along with many others) is the interface with applications running in different processes, communicating with the server using CGI, FastCGI, or SCGI protocols.

mod_wsgi vs. mod_proxy-based solution

I won't cover mod_wsgi in this talk. I currently use it for a couple of applications but am migrating away from it, primarily for these reasons:

- mod_proxy supports more separation between web server and application
    - Supports moving applications around or running applications in different modes for debugging without changing web server
    - Supports drastic 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 occasionally doesn't have releases for an extended period of time (e.g., required 2.4 users to collect patches for quite a while)

HTTP vs. FastCGI vs. SCGI

Further choices arise once mod_proxy is selected, the first of which is the wire protocol.

- 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.)

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 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

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

Applicable 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)
- any generic features added to mod_proxy in 2.4

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

I.e., improvements after, say, Ubuntu 14.04

Ubuntu 14.04 has 2.4.7; *current* is 2.4.12 or 2.4.13

- Unix socket support added in 2.4.10
- `CGIPassAuth` *to be added* in 2.4.13 or later
- maybe a redirect trick talked about here will be added *soon* too

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

Minimal build of httpd 2.4 to support Python applications

```
./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
```

Building blocks on the application side

- Django or Flask for the programming framework
- uWSGI for the “container” that hosts/manages the application processes
- an init script to start/stop the application by controlling uWSGI, and a uWSGI configuration file

Where is some of the sample code?

Later slides will show snippets from simple Flask and Django applications (and their server configurations) in the Github repository at `https://github.com/trawick/httpd.py`.

Some code was harmed in the development of this material!

- One topic in this presentation requires a mod_proxy_scgi patch to respect the use of the `X-Location` response header to control internal redirects from the application.
- This patch is in my httpd.py repository on Github.
- It needs to be generalized to support any custom header, not just `X-Location`, before proposing for a future 2.4.x release.

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

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

Simplest little bit of Flask

```
from flask import Flask

app = Flask(__name__)
@app.route('/app/cgivars/')

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

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

Listen 18082

<VirtualHost 127.0.0.1:18082>
    # Lots of stuff inherited from global scope
    SetEnvIf Request_URI . proxy-scgi-pathinfo
    ProxyPass / scgi://127.0.0.1:3005/
</VirtualHost>
```

Running the Flask app via uWSGI

```
VENV=/home/trawick/envs/httpd.py
${VENV}/bin/uwsgi --scgi-socket 127.0.0.1:3005 \
  --wsgi-file app.py \
  --callable app \
  --chdir /home/trawick/git/httpd.py/Flask \
  --virtualenv ${VENV}
```

Django: 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
```

Flask: X-Sendfile to offload file serving to the web server

```
from flask import Flask, request, send_file

app = Flask(__name__)
app.use_x_sendfile = True

@app.route('/app/sendfile/')
def sendfile():
    filename = request.environ['DOCUMENT_ROOT'] + '/' + 'bigfile.html'
    # This sets content-length to 0 so httpd sends 0 bytes from
    # the file.
    #
    # rsp = Response()
    # rsp.headers['X-Sendfile'] = filename
    # return rsp

    # This sets content-length from the actual file (and X-Sendfile).
    # It requires <app>.use_x_sendfile = True
    return send_file(filename)

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

Django: X-Location to offload request after application authorizes it

```
def protected(request):
    filename = '/static/protected/index.html'
    response = HttpResponse()
    # Django 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:
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>
```

Flask: X-Location to offload request after application authorizes it

```
@app.route('/app/protected/')
def protected():
    filename = '/static/protected/index.html'
    rsp = Response()

    # Flask/Werkzeug will turn this
    # into Location: http://127.0.0.1:18082/static/protected/foo
    #     rsp.headers['Location'] = '/protected/' + filename
    # This is passed through unadulterated:
    rsp.headers['X-Location'] = filename
    return rsp

# add to .conf for httpd:
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>
```

Handling /static/ for real Django apps

With the proper preparation, `./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`.

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.

Add load balancing

This is a fairly typical use of the load balancer; other talks at ApacheCon cover the load balancer capabilities more extensively.

```
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>
```

Handling Basic auth in the application

- Interactive applications normally use form+cookie-based auth.
- Basic auth handled by the application might be important for migration or other purposes.
- Normally httpd hides `Authorization` and `Proxy-Authorization` request headers from applications, but there are recipes on the web for subverting that, and mod_wsgi provides the `WSGIPassAuthorization` directive to enable that for applications it hosts.
- httpd 2.4.13 is expected to provide the `CGIPassAuth` directive to enable this for all CGI-like interfaces to applications, whether mod_fcgid, mod_wsgi, mod_cgi, mod_proxy extensions, or others.

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

Ansible-based configuration and deployment

We want something that deploys with a simple interface and handles many if not all aspects of system and application configuration.

```
$ ./deploy.sh staging

PLAY [Configure and deploy the application code] ******************************

GATHERING FACTS ***************************************************************
ok: [172.16.84.128]

TASK: [Install packages] ******************************************************
ok: [172.16.84.128] => (item=python-virtualenv,postgresql,libpq-dev,python-dev,python-psycopg2)

TASK: [Install git] ***********************************************************
ok: [172.16.84.128]

TASK: [Install git] ***********************************************************
skipping: [172.16.84.128]

TASK: [Install system httpd] **************************************************
ok: [172.16.84.128] => (item=apache2)

TASK: [Setup up Postgresql user] **********************************************
ok: [172.16.84.128]

TASK: [Setup up Postgresql DB] ************************************************
ok: [172.16.84.128]
```

Ansible-based configuration and deployment

```
TASK: [Add the logging group] *************************************************
ok: [172.16.84.128]

TASK: [Add managing user to logging group] ************************************
ok: [172.16.84.128]

TASK: [Add daemon user to logging group] **************************************
ok: [172.16.84.128]

TASK: [Create log directory] **************************************************
ok: [172.16.84.128]

TASK: [Create archive directory] **********************************************
ok: [172.16.84.128]

TASK: [git repo=ssh://git@github.com/trawick/{{ project_name }}.git dest={{ remote_checkout }} vers
changed: [172.16.84.128]

TASK: [template src={{ base_cfg_dir }}/settings.cfg.j2 dest={{ django_src }}/settings.cfg] ***
ok: [172.16.84.128]

TASK: [file dest={{ scratch_dir }} mode=755 owner={{ remote_user }} group={{ remote_user }} state=d
] ***
ok: [172.16.84.128]
```

Ansible-based configuration and deployment

```
TASK: [file dest={{ remote_checkout }}/envs mode=755 owner={{ remote_user }} group={{ remote_user ]
] ***
ok: [172.16.84.128]

TASK: [Create new virtualenv] *************************************************
skipping: [172.16.84.128]

TASK: [file dest={{ static_dir }} mode=755 owner={{ remote_user }} group={{ remote_user }} state=di
] ***
ok: [172.16.84.128]

TASK: [pip virtualenv={{ virtualenv_dir }} requirements={{ remote_checkout }}/src/requirements.txt]
ok: [172.16.84.128]

TASK: [django_manage app_path={{ django_src }} command=migrate virtualenv={{ virtualenv_dir }}
] ***
ok: [172.16.84.128]

TASK: [django_manage app_path={{ django_src }} command=collectstatic virtualenv={{ virtualenv_dir ]
] ***
ok: [172.16.84.128]

TASK: [Create test data] ******************************************************
changed: [172.16.84.128]

TASK: [Define nightly_archive cron job] ***************************************
skipping: [172.16.84.128]
```

Ansible-based configuration and deployment

```
TASK: [Configure system httpd to include mod_proxy] **************************
ok: [172.16.84.128]

TASK: [Configure system httpd to include mod_proxy_scgi] *********************
ok: [172.16.84.128]

TASK: [Configure system httpd to include mod_headers] ************************
ok: [172.16.84.128]

TASK: [Configure system httpd to include mod_deflate] ************************
ok: [172.16.84.128]

TASK: [Configure system httpd to include mod_expires] ************************
ok: [172.16.84.128]

TASK: [Configure system httpd] ***********************************************
ok: [172.16.84.128]

TASK: [Restart system httpd] *************************************************
changed: [172.16.84.128]

TASK: [Add application uWSGI config] *****************************************
ok: [172.16.84.128]
```

Ansible-based configuration and deployment

```
TASK: [Add application init script] *******************************************
ok: [172.16.84.128]

TASK: [Configure run-levels for application] **********************************
changed: [172.16.84.128]

TASK: [Restart application] ***************************************************
ok: [172.16.84.128]

PLAY RECAP ********************************************************************
172.16.84.128              : ok=31   changed=4    unreachable=0    failed=0
```

deploy.sh

```
$ cat deploy.sh
#!/bin/sh

usage="Usage: $0 {prod|staging}"
if test $# -ne 1; then
    echo $usage 1>&2
    exit 1
fi

if test $1 != "prod"; then
    if test $1 != "staging"; then
        echo $usage 1>&2
        exit 1
    fi
fi

. ~/envs/ansible/bin/activate
exec ansible-playbook -i $HOME/server-config/$1/walking/ansible-settings deploy.yml
```

deploy.yml - System packages

```
---

- name: Configure and deploy the application code
  hosts: webservers
  remote_user: "{{ remote_user }}"
  tasks:
    - name: Install packages
      apt: name={{ item }} state=latest
      sudo: yes
      with_items:
        - python-virtualenv
        - postgresql
        - libpq-dev
        - python-dev
# The system python-psycopg2 package is used by Ansible; the Django app uses psycopg2 from its virt
        - python-psycopg2

    - name: Install git
      apt: name=git state=latest
      sudo: yes

    - name: Install system httpd
      apt: name={{ item }} state=latest
      sudo: yes
      with_items:
        - apache2
```

deploy.yml - Database

```
    - name: Setup up Postgresql user
      sudo: yes
      sudo_user: postgres
      postgresql_user: name={{ pg_user }} password={{ pg_password }} \
role_attr_flags=CREATEDB,NOSUPERUSER

    - name: Setup up Postgresql DB
      sudo: yes
      sudo_user: postgres
      postgresql_db: name={{ project_db }}
                     encoding='UTF-8'
```

deploy.yml - Updating application from git

```
- git: repo=ssh://git@github.com/trawick/{{ project_name }}.git
       dest={{ remote_checkout }}
       version=HEAD
       update=yes
       force=no
       key_file=/home/{{ remote_user }}/.ssh/{{ git_deploy_key }}
```

deploy.yml - virtualenv

```
    - file: >
          dest={{ remote_checkout }}/envs
          mode=755
          owner={{ remote_user }}
          group={{ remote_user }}
          state=directory

    - name: Create new virtualenv
      command: "{{ virtualenv_binary }} -p {{ python_binary }} \
--no-site-packages {{ virtualenv_dir }} creates={{ virtualenv_dir }}"

    - file: >
          dest={{ static_dir }}
          mode=755
          owner={{ remote_user }}
          group={{ remote_user }}
          state=directory

    - pip: virtualenv={{ virtualenv_dir }}
          requirements={{ remote_checkout }}/src/requirements.txt
```

deploy.yml - Django setup

```
- django_manage: >
      app_path={{ django_src }}
      command=migrate
      virtualenv={{ virtualenv_dir }}

- django_manage: >
      app_path={{ django_src }}
      command=collectstatic
      virtualenv={{ virtualenv_dir }}
```

deploy.yml - httpd configuration

```
    - name: Configure system httpd to include mod_proxy
      apache2_module: state=present name=proxy
      sudo: yes
    - name: Configure system httpd to include mod_proxy_scgi
      apache2_module: state=present name=proxy_scgi
      sudo: yes
    - name: Configure system httpd to include mod_headers
      apache2_module: state=present name=headers
      sudo: yes
    - name: Configure system httpd to include mod_deflate
      apache2_module: state=present name=deflate
      sudo: yes
    - name: Configure system httpd to include mod_expires
      apache2_module: state=present name=expires
      sudo: yes
    - name: Configure system httpd
      template: src={{ base_cfg_dir }}/ubuntu-apache24/{{ project_name }}-vhost.conf \
dest=/etc/apache2/sites-enabled/
      sudo: yes
    - name: Restart system httpd
      command: /etc/init.d/apache2 reload
      sudo: yes
```

deploy.yml - uWSGI configuration

```
- name: Add application uWSGI config
  template: src=uwsgi-ini.j2 dest={{ log_dir }}/{{ project_name }}.ini

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

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

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

deploy.yml - .conf template

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

<VirtualHost *:443>
    ServerName {{ canonical_server_name }}

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

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

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

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

deploy.yml - .conf template

```
    # plain "SetEnv" sets this too late
    SetEnvIf Request_URI . proxy-scgi-pathinfo

    ProxyPass /robots.txt !
    ProxyPass /static/ !
   # ProxyPass /media/ !
    ProxyPass / scgi://127.0.0.1:{{ application_port }}/

    <Location /admin/>
<IfModule ssl_module>
        Require ssl
</IfModule>
    </Location>

    <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>
...
```

deploy.yml - .conf template

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

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

    SSLCertificateKeyFile   /home/trawick/server_keys/arewewalkingtomorrow.com/arewewalkingtomorro
    SSLCertificateFile      /home/trawick/server_keys/arewewalkingtomorrow.com/arewewalkingtomorro
    SSLCertificateChainFile /home/trawick/server_keys/arewewalkingtomorrow.com/all.pem
</VirtualHost>
```

deploy.yml - uWSGI template

```
[uwsgi]
pidfile = {{ log_dir }}/{{ project_name }}.pid
daemonize = {{ log_dir }}/{{ project_name }}.log
scgi-socket = 127.0.0.1:{{ application_port }}
chdir = {{ django_src }}
module = {{ project_name }}.wsgi
master = true
processes = 1
threads = 2
uid = {{ remote_user }}
gid = {{ remote_user }}
virtualenv = {{ virtualenv_dir }}
```

deploy.yml - init script

```
!/bin/sh

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

. ${UWSGI_ENV}/bin/activate

start_service() {
    if test -f "$PIDFILE"; then
        echo " * $SERVICE_NAME pid file already exists..."
        PID=`cat $PIDFILE`
        if kill -0 $PID 2>/dev/null; then
            echo " * $SERVICE_NAME is already running"
            exit 1
        fi
...
```

(and on and on)

Here's a complete example: https://github.com/trawick/edurepo/blob/master/src/ansible/init-script.j2

deploy.yml - Template variables

a.k.a. Ansible hosts file

```
[webservers]
# This is the IP address or hostname of the server machine.
arewewalkingtomorrow.com target_address=arewewalkingtomorrow.com \
canonical_server_name=arewewalkingtomorrow.com \
canonical_base_url=http://arewewalkingtomorrow.com/

[webservers:vars]
base_cfg_dir=/home/trawick/server-config/prod/walking
application_port=3001
project_name=walking
remote_user=walker
remote_checkout=/home/{{ remote_user }}/git/{{ project_name }}
static_dir=/home/{{ remote_user }}/{{ project_name }}-static
httpd_log_level=warn
log_dir=/var/log/django-{{ project_name }}
project_db={{ project_name }}
pg_user={{ project_name }}
virtualenv_dir={{ remote_checkout }}/envs/{{ project_name }}
django_src={{ remote_checkout }}/src/{{ project_name }}
```

“pyweb”

`http://emptyhammock.com/projects/info/pyweb/index.html`

- Web Server Configuration for Python Apps, my *work-forever-in-progress* to describe similar httpd and nginx mechanisms for deploying Python applications
- Includes some performance comparisons, many more connectivity variations, etc.

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

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 time to first byte?[2]) 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

[2]`LogIOTrackTTFB` was just added to trunk; maybe it will be backported to 2.4.x soon.

Thank you!