Tutorial to seting up a django website in production.

gistfile1.rst

Set up Django, Nginx and Gunicorn in a Virtualenv controled by Supervisor

Steps with explanations to set up a server using:

- Virtualenv
- Virtualenvwrapper
- Django
- Gunicorn
- Nginx
- Supervisor

Concept

Nginx will face the outside world. It will serve media files (images, CSS, etc) directly from the file system. However, it can't talk directly to Django applications; it needs something that will run the application, feed it requests from the web, and return responses.

That's Gunicorn's job. Gunicorn will create a Unix socket, and serve responses to nginx via the wsgi protocol - the socket passes data in both directions:

```
The outside world <-> Nginx <-> The socket <-> Gunicorn
```

All this family will live into a Virtualenv. Already wondered why Virtualenv is so useful when you develop Pythons's applications? Continue to read and you will understand.

Before you start

1. Virtualenv

If you don't already use a Virtualenv, you really should consider using it. In a nutshell, the fact is that your operating system have a lot of core modules which depends of Python. That rule is especially right if you use Ubuntu. So, do you really need to broke up something existant to create somethin new? Well, some guys could answer you "Yes". I'll teach you how to say "No" to them ! Virtualenv will create a dedicated virtual environment with his python binary as well as his own modules. With that method, your Virtualenv will be isolated from the rest of the system.

1.5 Pip (optional, but higly recommanded)

If you use Pip, and I clearly recommand to use it, Virtualenv will simplify your life, because if you play well, you only have to do

```
pip freeze > requirements.txt
```

to print a list of all the modules needed by your project into a file. When you will deploy it, you will be able to install all those module with a

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

2. Virtualenvwrapper

What ? I didn't explained how to use Virtualenv? Well, indeed I lied. We won't be using Virtualenv directly. We will use a wrapper. Virtualenvwrapper to be right. Original, isn't it? Virtualenvwrapper will allow you to easily create virtualenv and switching between them.

Let's get started ! To install virtualenvwrapper, install it with Pip, apt-get, whatever:

```
pip install virtualenvwrapper
```

Virtualenvwrapper is now installed? Let's play with him. First, you will need to modify your .bashrc with your favourite text editor. Add this lines at the and of your .bashrc

```
export WORKON_HOME=~/.virtualenvs
export PROJECT_HOME=/path/to/your/project/home
source /usr/local/bin/virtualenvwrapper.sh
```

WORKON_HOME is required but PROJECT_HOME is needless. Don't forget to always put ALL the export's directives before sourcing your file. Seems legit for you? Here, have my like !

Now, let's activate our .bashrc. If you are a UNIX guru, you have problably already did it before I explained it. If not, here is the command

```
source .bashrc
```

If all is good, your stdout will be sourced byt the output of the Virtualenvwrapper creating script. You can verify it by searching a .virtualenvs folder is your home directory.

Now, let's ceate our virtualenv for our project. You can name it whatever your want and even if you don't remember, just go into your .virtualenvs folder and the name of your virtualenv will be one of the folder. The command for creating one is

```
mkvirtualenv nameofyourproject
```

The command for working on a specified virtualenv is

```
workon nameofyourproject
```

The command for deactivate a specified virtualenv is

```
deactivate nameofyourproject
```

And finally the command to remove a specified virtualenv is

```
rmvirtualenv nameofyourproject
```

That's all the command you need to remember. If the workon command has successfully created a virtualenv, your should see the name of your project between parentheses in the left of your username@nameofyourcomputer. Something like that

```
(nameofyourproject)username@nameofyourcomputer
```

All is good, Okey, let's move to the next of the story.

! REMEMBER TO DO ALL THE NEXT PART IN YOUR VIRTUALENV. ONLY SUPERVISOR WILL BE INSTALLED OUTSIDE YOUR VIRTUALENV !

1. Django

I'm assuming you are using Django 1.4.x. Everything should be fine from our right for Django 1.5. Anayway, if you created your project in 1.4, it should have automatically created a wsgi module. If you're using an earlier version, you will have to find a Django wsgi module for your project. Here is mine modified for exemple

```
import os
import sys


root = os.path.join(os.path.dirname(__file__), '..')
sys.path.insert(0, root)


os.environ['DJANGO_SETTINGS_MODULE'] = 'nameofyourproject.settings'


import django.core.handlers.wsgi
application = django.core.handlers.wsgi.WSGIHandler()

# This application object is used by any WSGI server configured to use this
# file. This includes Django's development server, if the WSGI_APPLICATION
# setting points here.
from django.core.wsgi import get_wsgi_application
application = get_wsgi_application()

# Apply WSGI middleware here.
# from helloworld.wsgi import HelloWorldApplication
# application = HelloWorldApplication(application)
```

Note that I'm also assuming a Django 1.4 project structure, in which you see paths like:

```
/path/to/your/project/project/
```

(i.e. it creates nested directories with the name of your project). Adjust the examples if you using an earlier Django.

It will also be helpful if you are in your Django project's directory. If you don't have one ready, just create a directory for now.

About the domain and port

I'll call your domain domain.tld. Substitute your own FQDN or IP address.

Throughout, I'm using the port (:8000) for tests because it is the same port as Django's dev web server. The default port for http (:80) will be used for real deployment. You can use whatever port you want of course, but if you don't know what NAT is, don't do it.

Basic Gunicorn installation and configuration

Install Gunicorn

As we said earlier, Gunicorn will serve the core of our application, just like the Django's development web server would do it. Let's install it

```
pip install gunicorn
```

Basic test

Create a file called myapp.py:

```
def app(environ, start_response):
    data = "Hello, World!\n"
    start_response("200 OK", [
        ("Content-Type", "text/plain"),
        ("Content-Length", str(len(data)))
    ])
return iter([data])
```

Run:

```
gunicorn -w 4 myapp:app
```

This should serve a hello world message directly to the browser on port 8000. Visit:

```
http://127.0.0.1:8000
```

to check.

Test your Django project

Now we want gunicorn to do the same thing, but to run a Django site instead of the test.py module.

But first, make sure that your project actually works! Now you need to be in your Django project directory.

```
python manage.py runserver 0.0.0.0:8000
```

Now run it using gunicorn:

```
gunicorn nameofyourapp.wsgi:app
```

Point your browser at the server; if the site appears, it means gunicorn can serve your Django application from your virtualenv. Media/static files may not be served properly, but don't worry about that.

Now normally we won't have the browser speaking directly to gunicorn: nginx will be the go-between.

Basic nginx

Install nginx

The version of Nginx from Debian stable and Ubuntu is rather old. We'll install from backports for Debian and PPA for Ubuntu.

Debian:

```
sudo nano /etc/apt/sources.list     # edit the sources list
```

Add:

```
# backports
deb http://backports.debian.org/debian-backports squeeze-backports main
```

Run:

```
sudo apt-get -t squeeze-backports install nginx # install nginx
```

For Ubuntu:

```
sudo -s
nginx=stable # use nginx=development for latest development version
add-apt-repository ppa:nginx/$nginx
apt-get update
apt-get install nginx
```

For both:

```
sudo /etc/init.d/nginx start    # start nginx
```

And now check that the server is serving by visiting it in a web browser on port 80 - you should get a message from nginx: "Welcome to nginx!"

Configure nginx for your site

Create a file called nginx.conf, and put this in it:

```
server {
    # the port your site will be served on
    listen      80;
    # the domain name it will serve for
    server_name .domain.tld ip.adress;   # substitute by your FQDN and machine's IP address
    charset     utf-8;

    #Max upload size
    client_max_body_size 75M;   # adjust to taste

    # Django media
    location /media  {
        alias /var/www/path/to/your/project/media;      # your Django project's media files
    }

    location /assets {
        alias /var/www/path/to/your/project/static;     # your Django project's static files
    }

    # Finally, send all non-media requests to the Django server.
    location / {
        proxy_pass http://127.0.0.1:8000;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    }
}
```

Move your file to /etc/nginx/sites-enabled so nginx can see it:

```
sudo mv /path/to/your/project/nginx.conf /etc/nginx/sites-enabled/
```

Basic nginx test

Restart nginx:

```
sudo /etc/init.d/nginx restart
```

Check that media files are being served correctly:

Add an image called media.png to the /path/to/your/project/project/media directory

Visit

http://domain.tld:80/media/media.png

If this works, you'll know at least that nginx is serving files correctly.

Running the Django application with uswgi and nginx

Let's run our Django application:

```
gunicorn nameofyourapp.wsgi:app
```

Now gunicorn and nginx should be serving up your Django application.

Make gunicorn startup when the system boots

The last step is to make it all happen automatically at system startup time.

To do this, we will install a programm called supervisor:

```
sudo pip install supervisor
```

Sudo in front of your command is important, because it means that our supervisor programm will be installed in the system and not in a virtualenv.

Now, we will create a bash file to execute some commands such as activate our virtualenv and browsing to our directory. So create a nameoryourproject.sh file:

```
#!/bin/bash
WORKING_DIR=/path/to/your/project
ACTIVATE_PATH=/path/to/your/virtualenv/bin/activate
cd ${WORKING_DIR}
source ${ACTIVATE_PATH}
exec $@
```

Next, we create a file named supervisord.conf and we put it in /etc/

```
; Sample supervisor config file.
;
; For more information on the config file, please see:
; http://supervisord.org/configuration.html
;
; Note: shell expansion ("~" or "$HOME") is not supported.  Environment
```

```
; variables can be expanded using this syntax: "%(ENV_HOME)s".

[unix_http_server]
file=/tmp/supervisor.sock   ; (the path to the socket file)
chmod=0700                 ; socket file mode (default 0700)
chown=root:root       ; socket file uid:gid owner
;username=noname              ; (default is no username (open server))
;password=noname               ; (default is no password (open server))

;[inet_http_server]         ; inet (TCP) server disabled by default
;port=127.0.0.1:9001        ; (ip_address:port specifier, *:port for all iface)
;username=noname              ; (default is no username (open server))
;password=noname               ; (default is no password (open server))

[supervisord]
logfile=/var/log/supervisord/supervisord.log ; (main log file;default $CWD/supervisord.log)
logfile_maxbytes=50MB        ; (max main logfile bytes b4 rotation;default 50MB)
logfile_backups=10           ; (num of main logfile rotation backups;default 10)
loglevel=info                ; (log level;default info; others: debug,warn,trace)
pidfile=/tmp/supervisord.pid ; (supervisord pidfile;default supervisord.pid)
nodaemon=true               ; (start in foreground if true;default false)
minfds=1024                  ; (min. avail startup file descriptors;default 1024)
minprocs=200                 ; (min. avail process descriptors;default 200)
;umask=022                   ; (process file creation umask;default 022)
user=root                 ; (default is current user, required if root)
;identifier=supervisor       ; (supervisord identifier, default is 'supervisor')
;directory=/tmp              ; (default is not to cd during start)
;nocleanup=true              ; (don't clean up tempfiles at start;default false)
;childlogdir=/tmp            ; ('AUTO' child log dir, default $TEMP)
;environment=KEY=value       ; (key value pairs to add to environment)
;strip_ansi=false            ; (strip ansi escape codes in logs; def. false)

; the below section must remain in the config file for RPC
; (supervisorctl/web interface) to work, additional interfaces may be
; added by defining them in separate rpcinterface: sections
[rpcinterface:supervisor]
supervisor.rpcinterface_factory = supervisor.rpcinterface:make_main_rpcinterface

[supervisorctl]
serverurl=unix:///tmp/supervisor.sock ; use a unix:// URL  for a unix socket
;serverurl=http://127.0.0.1:9001 ; use an http:// url to specify an inet socket
;username=www-data              ; should be same as http_username if set
;password=www-data                ; should be same as http_password if set
;prompt=mysupervisor         ; cmd line prompt (default "supervisor")
;history_file=~/.sc_history  ; use readline history if available

; The below sample program section shows all possible program subsection values,
; create one or more 'real' program: sections to be able to control them under
; supervisor.

;[program:theprogramname]
;command=/bin/cat              ; the program (relative uses PATH, can take args)
;process_name=%(program_name)s ; process_name expr (default %(program_name)s)
;numprocs=1                    ; number of processes copies to start (def 1)
;directory=/tmp                ; directory to cwd to before exec (def no cwd)
;umask=022                     ; umask for process (default None)
;priority=999                  ; the relative start priority (default 999)
;autostart=true                ; start at supervisord start (default: true)
;autorestart=unexpected        ; whether/when to restart (default: unexpected)
;startsecs=1                   ; number of secs prog must stay running (def. 1)
;startretries=3                ; max # of serial start failures (default 3)
;exitcodes=0,2                 ; 'expected' exit codes for process (default 0,2)
;stopsignal=QUIT               ; signal used to kill process (default TERM)
;stopwaitsecs=10               ; max num secs to wait b4 SIGKILL (default 10)
;stopasgroup=false             ; send stop signal to the UNIX process group (default false)
;killasgroup=false             ; SIGKILL the UNIX process group (def false)
;user=chrism                   ; setuid to this UNIX account to run the program
```

```
;redirect_stderr=true          ; redirect proc stderr to stdout (default false)
;stdout_logfile=/a/path        ; stdout log path, NONE for none; default AUTO
;stdout_logfile_maxbytes=1MB   ; max # logfile bytes b4 rotation (default 50MB)
;stdout_logfile_backups=10     ; # of stdout logfile backups (default 10)
;stdout_capture_maxbytes=1MB   ; number of bytes in 'capturemode' (default 0)
;stdout_events_enabled=false   ; emit events on stdout writes (default false)
;stderr_logfile=/a/path        ; stderr log path, NONE for none; default AUTO
;stderr_logfile_maxbytes=1MB   ; max # logfile bytes b4 rotation (default 50MB)
;stderr_logfile_backups=10     ; # of stderr logfile backups (default 10)
;stderr_capture_maxbytes=1MB   ; number of bytes in 'capturemode' (default 0)
;stderr_events_enabled=false   ; emit events on stderr writes (default false)
;environment=A=1,B=2           ; process environment additions (def no adds)
;serverurl=AUTO                ; override serverurl computation (childutils)

; The below sample eventlistener section shows all possible
; eventlistener subsection values, create one or more 'real'
; eventlistener: sections to be able to handle event notifications
; sent by supervisor.

;[eventlistener:theeventlistenername]
;command=/bin/eventlistener    ; the program (relative uses PATH, can take args)
;process_name=%(program_name)s ; process_name expr (default %(program_name)s)
;numprocs=1                    ; number of processes copies to start (def 1)
;events=EVENT                  ; event notif. types to subscribe to (req'd)
;buffer_size=10                ; event buffer queue size (default 10)
;directory=/tmp                ; directory to cwd to before exec (def no cwd)
;umask=022                     ; umask for process (default None)
;priority=-1                   ; the relative start priority (default -1)
;autostart=true                ; start at supervisord start (default: true)
;autorestart=unexpected        ; whether/when to restart (default: unexpected)
;startsecs=1                   ; number of secs prog must stay running (def. 1)
;startretries=3                ; max # of serial start failures (default 3)
;exitcodes=0,2                 ; 'expected' exit codes for process (default 0,2)
;stopsignal=QUIT               ; signal used to kill process (default TERM)
;stopwaitsecs=10               ; max num secs to wait b4 SIGKILL (default 10)
;stopasgroup=false             ; send stop signal to the UNIX process group (default false)
;killasgroup=false             ; SIGKILL the UNIX process group (def false)
;user=chrism                   ; setuid to this UNIX account to run the program
;redirect_stderr=true          ; redirect proc stderr to stdout (default false)
;stdout_logfile=/a/path        ; stdout log path, NONE for none; default AUTO
;stdout_logfile_maxbytes=1MB   ; max # logfile bytes b4 rotation (default 50MB)
;stdout_logfile_backups=10     ; # of stdout logfile backups (default 10)
;stdout_events_enabled=false   ; emit events on stdout writes (default false)
;stderr_logfile=/a/path        ; stderr log path, NONE for none; default AUTO
;stderr_logfile_maxbytes=1MB   ; max # logfile bytes b4 rotation (default 50MB)
;stderr_logfile_backups        ; # of stderr logfile backups (default 10)
;stderr_events_enabled=false   ; emit events on stderr writes (default false)
;environment=A=1,B=2           ; process environment additions
;serverurl=AUTO                ; override serverurl computation (childutils)

; The below sample group section shows all possible group values,
; create one or more 'real' group: sections to create "heterogeneous"
; process groups.

;[group:thegroupname]
;programs=progname1,progname2  ; each refers to 'x' in [program:x] definitions
;priority=999                  ; the relative start priority (default 999)

; The [include] section can just contain the "files" setting.  This
; setting can list multiple files (separated by whitespace or
; newlines).  It can also contain wildcards.  The filenames are
; interpreted as relative to this file.  Included files *cannot*
; include files themselves.

;[include]
;files = relative/directory/*.ini
[program:nameofyourprogram]
```

```
directory = /path/of/your/project/
user = www-data
command = /path/to/your/nameofyourproject.sh gunicorn nameofyourproject.wsgi:application
stdout_logfile = /var/log/supervisord/access.log
stderr_logfile = /var/log/supervisord/error.log
```

After, we will create a file which will ask supervisord to boot at the startup.

Create a file named supervisord:

```
# Supervisord auto-start
#
# description: Auto-starts supervisord
# processname: supervisord
# pidfile: /var/run/supervisord.pid

SUPERVISORD=/usr/local/bin/supervisord
SUPERVISORD_ARGS='-c /etc/supervisord.conf'
SUPERVISORCTL=/usr/local/bin/supervisorctl

case $1 in
start)
    echo -n "Starting supervisord: "
    $SUPERVISORD $SUPERVISORD_ARGS
    echo
    ;;
stop)
    echo -n "Stopping supervisord: "
    $SUPERVISORCTL shutdown
    echo
    ;;
restart)
    echo -n "Stopping supervisord: "
    $SUPERVISORCTL shutdown
    echo
    echo -n "Starting supervisord: "
    $SUPERVISORD $SUPERVISORD_ARGS
    echo
    ;;
esac
```

Make it executable

```
chmod +x supervisord
```

Move it to /etc/init.d

```
mv supervisord /etc/init.d/
```

Make it boot at the start

```
sudo update-rc.d supervisord defaults
```

Reboot your system and your website should live through the ages.