Practical Python Web Deployments

James Polera
james@uncryptic.com // @uncryptic
Python Users Group in Princeton
02.13.2012

About me

* My name is James Polera, I'm the IT manager for a mid-size law firm in Union county, and part of the sister technology company (they do .NET primarily). I also own a consulting company where I do mostly Python/Django apps (in my spare time, of which I have none).

* I've been working in IT professionally since 2000 in both Sysadmin and Developer roles (often at the SAME TIME).

About this talk

* What I'm about to present to you are tools and methods that I have used, and services that I think make things easy and fun.

* This is by no means a definitive guide to deploying web applications. All applications are different, and I'm sure yours has a need that we won't touch on. I'd like to make this as informative as possible, so when we get to the end please bring up your use case and share what you did to accomplish the task at hand.

* If you think I'm wrong about anything, tell me. I'd rather know if there is a better way to do things.

Things we won't cover

* Operating System configuration
* Firewall configuration
* Refrigerator repair
* Database configuration (not *every* project has one)
* Installing modules via pip

Developing a foundation
Source Control

* One of the first things I do when starting any project is to layout a code repository.

* Using source control is technically a decision (and you should decide to use it).

Developing a foundation
Environmental control

* How many of you have heard of virtualenv?
* How many of you use virtualenv?
* Sure, but how do I use it?

Developing a foundation
Environmental control

* I recommend installing both **virtualenv** and **virtualenvwrapper** (and a module named **yolk** – yolk allows you to list the installed modules in an environment and their versions)
    * export WORKON_HOME=”~/.virtualenvs”
    * mkdir -p $WORKON_HOME
    * source /usr/local/bin/virtualenvwrapper.sh
    * mkvirtualenv myenv1
* Now, you can install whatever modules you want in your virtual environment without affecting your base install. (leave your virtualenv with deactivate, jump into your virtualenv with workon ENV_NAME_HERE)

Developing a foundation
Environmental control

* You might be thinking, “Hey, I want to learn more about virtualenv”

* I didn't prepare for that.

* That *does* sound like a great idea for a lightning talk / presentation.

* I hope somebody does it!

Developing a foundation
Environmental control

* You're probably asking, so what does all this have to do with deployments?

* Something I didn't touch on with virtualenv was the ability to “freeze” your virtual environment.

* pip freeze -l > requirements.txt (from within your virtualenv)

* Having a requirements.txt file is a great way to ensure that your app works the same in production as it does in development.

Developing a foundation
Summary

* Source control and environment management allow you to develop fearlessly, but as the saying goes – shipping is the most important feature.

* At this point you've come to a crossroad and it's time to make a decision. You have your product, you're satisfied with it, it's tagged for deployment in your revision control system. You're ready to ship.

    * I'm sure that during the development process (and hopefully before, for that matter) you at least determined what your system requirements were going to be (i.e. Windows, Linux, BSD)

    * Maybe you didn't. I think this is rare, but I suppose it could happen if you're working on a personal project. You take your idea and implement it without consideration for your production environment.

* FEAR NOT: we're going to look at a few.

Sidebar – Static content

* Have a plan for this.
* Your plan should be to either:
    * Use a CDN (like Amazon S3 or Rackspace Cloud Files)
    * Alias your static content directory in your web server config. (this is probably the most common solution)
* Don't serve your static content through Python! (you're going to have a bad time – trust me)

Deployment – (yay, we finally got here!)

* The current tech landscape offers many options
* Unmanaged PAAS (Amazon Web Services (EC2), Linode, Rackspace Cloud, RootBSD, VPS Networks, etc.)
* Managed PAAS (your Heroku, DotCloud, etc.)
* Shared hosting (Webfaction, etc.)

Deployment – (yay, we finally got here!)

* Obviously each of these options have their pros and cons, depending upon your level of expertise.
    * DISCLAIMER: I was a sysadmin in a past life, and I remember enough to be dangerous :)
* The deployment scenarios we're going to look at are going to be WSGI (Web Server Gateway Interface) based. In the past, Python web applications dealt with CGI, FastCGI or mod_python, but WSGI really is the way to go these days (it's low-level and it is supported by many (if not all) of the common day web frameworks)
* We're going to use a simple Flask project to demo our deployment scenarios.

Deployment Scenario
Windows VPS
(or bare metal server)

* Let's get this one out of the way (kidding)
* Your job is to write Python code – because it's awesome – but you need to deploy in an environment that wasn't necessarily designed to run anything that wasn't ".Net compatible".
* Apache runs on Windows
* mod_wsgi is available for Apache on Windows
* At my company we've seen some serious threading issues with Django and mod_wsgi. I'd recommend deploying Python web apps to Windows only when it's absolutely required.
* I haven't put together a demo for this scenario since I don't have a Windows server to deploy to. If anyone is interested, I'll add an example Apache config to the Github repo (more on that later) under the "windows" branch. (side note: the Linux config won't be much different)

Hey! There's a middle layer for Linux or *BSD

* uWSGI
* What is it?
  * (From their website) - uWSGI is a fast, self-healing and developer/sysadmin-friendly application container server coded in pure C.
  * Python is the only language that has 100% uWSGI api support
  * It works with Virtualenv
  * It works with Apache, nginx, cherokee and lighttpd.
  * We're going to use it.

Deployment Scenario
Linux or *BSD
(VPS or bare metal)

* If you have any exposure to deploying web applications under Linux or *BSD in the past decade, you're probably familiar with deploying under Apache.
* As we said in the Windows deployment scenario, Apache supports a module named mod_wsgi.
* This module works on Linux and *BSD too.
* Let’s look at a config.
* http://wsgi.np.polera.org
*

Deployment Scenario
Linux or *BSD
(VPS or bare metal)

* Newsflash: Apache isn't the only kid in town anymore
  * Over the past few years, there has been a flurry of webserver development resulting in some fresh faces.
    * Nginx (http://www.nginx.com)
    * Cherokee Web Server (http://www.cherokee-project.com)
    * lighttpd (prounounced lighty) (http://www.lighttttpd.net)
    * I'm sure there are more, but we don't have all night.
  * I'm glad your wondering what these servers have in common.
  * At least two of the web servers that I mentioned above (nginx and cherokee) support uwsgi by default. All of them also work as excellent reverse proxy servers (if you wanted to run your app via FastCGI with any of these on the front end for example). Reverse proxying is kind of a sysadmin topic – if we want to expand on it and there are any sysadmins willing to do a lightning talk – I know I would appreciate it.
  * We're going to demo a deployment with nginx (hey, it's probably the most popular of the ones I listed above. You want to go home tonight, don't you?)
    * http://uwsgi.np.polera.org

Deployment Scenario Heroku

* Let's say that you don't want to be sysadmin and developer.
* Maybe you want to “just write code” and deploy
  * Enter Heroku
    * This is the part where you wish you had been using git all along.
      * Heroku's deployment process is git
      * You setup your app.
      * You create your requirements.txt file
      * You add a git remote repository for Heroku
      * You push to that repository
      * Profit (DISCLAIMER: you may not profit)
* Let’s look at our last demo http://heroku.np.polera.org

Conclusion

* I'm not going to go over the shared hosting scenario because you don't have as much control in a shared hosting environment. In my opinion (throw things if you must) if you're looking to go with a shared hosting setup – I'd recommend looking to PAAS providers like Heroku. You get a lot for a little (or even free!)
    * Things I don't know much about, but would like to learn (hint, hint):
        * Deploying Python with Python (I hear you can deploy things like Flask with Tornado)
        * Deploying any of this stuff with PyPy
        * I'm sure there's something I'm forgetting.
    * **I hope you enjoyed this and I really hope that you learned something from it.**
    * Comments/Questions?

Some relevant links (in order of appearance)

* git: http://git-scm.com/
* pip: http://pypi.python.org/pypi/pip
* virtualenv: http://pypi.python.org/pypi/virtualenv
* virtualenvwrapper: http://www.doughellmann.com/projects/virtualenvwrapper
* EC2 and S3: https://aws.amazon.com/
* Rackspace Cloud: http://www.rackspace.com/cloud/
* Linode: http://www.linode.com/
* Heroku: http://www.heroku.com/
* mod_wsgi: http://code.google.com/p/modwsgi/
* uwsgi: http://projects.unbit.it/uwsgi/

More relevant links (not in order of appearance)

* Apache: http://httpd.apache.org/
* nginx: http://wiki.nginx.org/Main
* Cherokee Web Server: http://www.cherokee-project.com/
* Source code: https://github.com/polera/practical_python_deployments