新手的Flask+uwsgi+Nginx+Ubuntu部署过程

Cocoa_Coder 关注
0.948 2016.11.17 09:15:33 字数 5,086 阅读 7,058

学习 Flask，写完一个 Flask 应用需要部署的时候，就想着折腾自己的服务器。根据搜索的教程照做，对于原理一知半解，磕磕碰碰，只要运行起来了，谢天谢地然后不再折腾了，到下一次还需要部署时，这样的过程就会重复一次。不知道多少人的膝盖中箭了呢？我也这样干过，这么做确实很蠢，所以我决定写一篇 Flask+uwsgi+Nginx+Ubuntu 的部署教程，解答一些我自己在这个过程中的疑问，从原理到方案，以一个小白的角度，总结一下部署、运维这件事，应该对初学 Flask 需要部署的同学有些帮助。

more

环境简介

Ubuntu

我使用的 Ubuntu 系统版本是 14.04，用过几个 Linux 发行版，现在挑选系统的第一选择基本就是 Ubuntu 了，因为 Ubuntu 有商业公司 Canonical 做开发维护；使用的人多，有庞大的社区支持；遇到问题容易解决。我折腾过很长时间的 Linux 系统，我对新手的建议是，不要把时间浪费在这上面，应该以解决实际问题为导向，踏实点提高编程能力。装系统、优化系统、记各种酷炫的命令对于提高编程能力并没有实际帮助。所以你问我资瓷不资瓷 Ubuntu，我当然是资瓷的啦，用 Ubuntu 当然也会遇到坑，但相比于其他系统会少一些，也会容易解决一点。事实上，Ubuntu 已经成为了服务器的首选，AWS 上被选择最多的 Linux 发行版就是 Ubuntu。Quora 用的 Linux 发行版也是 Ubuntu，创始人 Adam D'Angelo 在这个回答里解释了原因。总的来说，没有特别的理由的话，Ubuntu 理应是首选，经验多一些之后，如果对某个发行版感兴趣，或者想要做一些特别的尝试，跳出舒适区，试试其他系统也无妨。

uWSGI

我们知道 Flask 中自带了 web server，通过 Werkzeug，我们可以搭建 WSGI 服务，运行我们的网站，但 Flask 是 Web 框架，并不是 Web 服务器，尽管 Werkzeug 很强大，但只能用于开发，不能用于生产，对于 Web 服务器，我们有更专业的选择，那就是 uWSGI, uWSGI 是一个全站式的托管服务，它实现了应用服务器（支持多种编程语言）、代理、进程管理器、监视器。取名为 uWSGI 是因为它最早实现的是 Python 语言的 WSGI。

uWSGI 包括四个部分：

- uwsgi协议
- web server 内置支持协议模块
- application 服务器协议支持模块
- 进程控制程序

uWSGI 是 C 语言写的，性能比较高。

WSGI, uWSGI, uwsgi 的区别

当我们部署完一个应用程序，浏览网页时具体的过程是怎样的呢？首先我们得有一个 Web 服务器来处理 HTTP 协议的内容，Web 服务器获得客户端的请求，交给应用程序，应用程序处理完，返回给 Web 服务器，这时 Web 服务器再返回给客户端。Web 服务器与应用程序之间显然要进行交互，这时就出现了很多 Web 服务器与应用程序之间交互的规范，最早出现的是 CGI，后来又出现了改进 CGI 性能的FasgCGI，Java 专用的 Servlet 规范，Python 专用的 WSGI 规范等等。有了统一标准，程序的可移植性就大大提高了。这里我们只介绍 WSGI。

WSGI 全称是 Web Server Gateway Interface，也就是 Web 服务器网关接口，它是 Python 语言定义出来的 Web 服务器和 Web 应用程序之间的简单而通用的接口，基于现存的 CGI 标准设计，后来在很多其他语言中也出现了类似的接口。 总的来说，WSGI 可以分为服务器和应用程序两个部分，实际上可以将 WSGI 理解为服务器与应用程序之间的一座桥，桥的一边是服务器，另一边是应用程序。

推荐阅读

一对"不知羞耻"的母女，撕开了我们致命的痛点，这片让人深思
阅读 11,769

Ab们再不努力，就要被杨超越吊打了
阅读 6,925

《中餐厅》黄晓明被骂的背后，藏着一个杠精诞生的底层逻辑
阅读 22,245

韩信为什么活埋了母亲?
阅读 21,199

开发部署提速8倍！这款IDE插件了解一下?
阅读 8,351

的部分就要复杂一点，可以通过 uWSGI 实现，也可以用最常见的 Web 服务器，比如 Apache、Nginx，但这些 Web 服务器没有内置 WSGI 的实现，是通过扩展完成的。如 Apache，通过扩展模块 mod_wsgi 来支持WSGI，Nginx可以通过代理的方式，将请求封装好，交给应用服务器，比如 uWSGI。uWSGI 可以完成 WSGI 的服务端，进程管理以及对应用的调用。WSGI 中间件的部分可以这样理解：我们把 WSGI 看做桥，这个桥有两个桥墩，一个是应用程序端，另一个是服务器端，那么桥面就是 WSGI 中间件，中间件同时具备服务器、应用程序端两个角色，当然也需要同时遵守 WSGI 服务器和 WSGI 应用程序两边的限制和需要。更详细的内容可以看PEP-333 中间件的描述

Flask 依赖的 Werkzeug 就是一个 WSGI 工具包，官方文档的定义是 Werkzeug 是为 Python 设计的 HTTP和 WSGI 实用程序库。我们需要注意的是，Flask 自带的 Werkzeug 是用来开发的，并不能用于生产环境，Flask 是 Web 框架，而 Werkzeug 不是 Web框架，不是 Web 服务器，它只是一个 WSGI 工具包，它在 Flask 的作用是作为 Web 框架的底层库，它方便了我们的开发。

我们将 uwsgi 和 uWSGI 放在一起讲解。uWSGI 是一个 Web 服务器程序，WSGI，上面已经谈到，是一种协议，uwsgi 也是一种协议，uWSGI 实现了 uwsgi、WSGI、http 等协议。 uwsgi 的介绍可以看这里，uwsgi 是 uWSGI 使用的一个自有的协议，它用4个字节来定义传输数据类型描述。尽管都是协议，uwsgi 和 WSGI 并没有联系，我们需要区分这两个词。

Nginx

Nginx 是高效的 Web 服务器和反向代理服务器，可以用作负载均衡（当有 n 个用户访问服务器时，可以实现分流，分担服务器的压力），与 Apache 相比，Nginx 支持高并发，可以支持百万级的 TCP 连接，十万级别的并发连接，部署简单，内存消耗少，成本低，但 Nginx 的模块没有 Apache 丰富。Nginx 支持 uWSGI 的 uwsgi 协议，因此我们可以将 Nginx 与 uWSGI 结合起来，Nginx 通过 uwsgi_pass 将动态内容交给 uWSGI 处理。

uWSGI 和 Nginx 的关系

从上面的讲解中，我们知道，uWSGI 可以起到 Web 服务器的作用，那么为什么有了 uWSGI 还需要 Nginx 呢?

最普遍的说法是 Nginx 对于处理静态文件更有优势，性能更好。其实如果是小网站，没有静态文件需要处理，只用 uWSGI 也是可以的，但加上 Nginx 这一层，优势可以很具体：

1 对于运维来说比较方便，如果服务器被某个 IP 攻击，在 Nginx 配置文件黑名单中添加这个 IP 即可，如果只用 uWSGI，那么就需要在代码中修改了。另一方面，Nginx 是身经百战的 Web 服务器了，在表现上 uWSGI 显得更专业，比如说 uWSGI 在早期版本里是不支持 https 的，可以说 Nginx 更安全。

2 Nginx 的特点是能够做负载均衡和 HTTP 缓存，如果不止一台服务器，Nginx 基本就是必选项了，通过 Nginx，将资源可以分配给不同的服务器节点，只有一台服务器，也能很好地提高性能，因为 Nginx 可以通过 headers 的Expires or E-Tag，gzip 压缩等方式很好地处理静态资源，毕竟是 C 语言写的，调用的是 native 的函数，针对 I/O做了优化，对于动态资源来说，Nginx 还可以实现缓存的功能，配合 CDN 优化（这是 uWSGI 做不到的）。Nginx 支持epoll/kqueue 等高效网络库，能够很好地处理高并发短连接请求，性能比 uWSGI 不知道高到哪里去了。

3 如果服务器主机上运行了PHP，Python 等语言写的多个应用，都需要监听80端口，这时候 Nginx 就是必选项了。因为我们需要一个转发的服务。

所以说，Nginx 基本也是必选项。

部署准备工作

这里我假设我们拿到的是一台全新的服务器。 一般来说，Linux 系统都会预装 Python 的，但不一定装了 easy_install 工具，我们可以通过 apt-get install python-setuptools 来安装 easy_install，再通过 easy_install 安装 pip。

搞定 Python 环境

```
$ sudo apt-get install python-setuptools
$ sudo easy_install pip
```

这样，我们就可以通过 pip 安装 virtualenv，为 flask 项目构建虚拟环境。
VirtualEnv
不同的项目可能会引用各种不同的依赖包，为了避免版本与和应用之间的冲突而造成的"依赖地狱"
[Virtualenv | https://virtualenv.readthedocs.org/en/latest/] 就是我们python 项目的必须品了。
VirtualEnv 可以为每个Python应用创建独立的开发环境，使他们互不影响，Virtualenv 能够做到：
• 在没有权限的情况下安装新套件
• 不同应用可以使用不同的套件版本
• 套件升级不影响其他应用
安装：

```
sudo pip install virtualenv
```

安装VirtualEnv 后只需要在项目目录内运行 virtualenv 目录名 就可以建立一个虚拟环境文件夹，然后启用 activate 指令即可启用该python虚拟环境，具体操作如下：
假定我的项目目录叫 /home/www/my_flask ，首先安装虚拟环境 (我习惯使用的虚拟环境目录叫 venv)

```
my_flask root$ virtualenv venv

>> New python executable in venv/bin/python
>> Installing setuptools, pip...done.
```

在项目目录下就会建立一个新的 venv 目录，里面就是运行python 的基本环境的工具与指令，和包。 然后启用该环境，使用当前命令行状态进入虚拟环境，进入虚拟环境后，一切安装python的操作都会将包和引用装在虚拟环境内，而不会影响到全局的python 环境。

```
my_flask root$ source venv/bin/activate （进入虚拟环境）
```

(venv)my_flask root$

调用 activate 指令后命令符前就会出现 (venv) 字样。 可通过 deactivate 退出虚拟环境。
安装 uWSGI
Flask 的实际生产运行环境选择并不多，比较成熟的是 [Gunicorn| http://gunicorn.org/] 和 [uWSGI| https://github.com/unbit/uwsgi]，听说Gunicorn 的配置很简单，但可惜我一直没有配置成功过，
在安装 uWSGI 前，需要解决 uWSGI 的依赖问题，因为 uWSGI 是一个 C 语言写的应用，所以我们需要 C 编译器，以及 python 开发相关组件：

```
$ sudo apt-get install build-essential python-dev（这是两个包）
```

我现采用的是 uWSGI。接下来就安装uWSGI吧。

```
(venv)my_flask root$ pip install uwsgi
```

到这，我们就安装好了 uWSGI，

```
(venv)my_flask root$ pip install flask
```

可能由于服务器原因 一直安装失败，请重新安装，所需依赖包可以自动安装，不用管它

安装Nginx

```
$ sudo apt-get install nginx
```

启动 nginx 的方法：

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

这时候在浏览器地址栏输入服务器的 ip 地址，看到下面的页面就表明 Nginx 已经启动了：

开干
首先，我们把应用程序上传到服务器中，我在用 git 管理项目，所以只需要 git clone 一下就可以了：
$ git clone http://url/of/you/git/repo
如果你需要从本地上传项目文件，可以用 scp 命令，这里就不啰嗦用法了。总之我们将项目文件放到服务器，然后就可以用 virtualenv 管理 Python 环境：

```
$ virtualenv ENV
$ source ENV/bin/activate         # 激活虚拟环境
$ pip install -r requirement.txt  # 解决依赖问题
$ deactivate                              # 退出依赖环境
```

这里就用 Flask 的7行代码做示例吧，我新建了一个文件夹，名为 helloflask，将下面的内容：

```
from flask import Flask
app = Flask(__name__)

@app.route("/")
def hello():
    return "Hello World!"

if __name__ == "__main__":
    app.run(host='0.0.0.0', port=5001)
```

保存为hello.py，运行试试，在浏览器输入服务器本机地址（192.168.1.X），加端口号5001就可以看到结果。
好了，现在我们用 Nginx 来承担 Web 服务。
修改 Nginx 的默认配置文件：
文本编辑器打开 /etc/nginx/sites-enabled/default 文件
有心的话，其实可以从 Nginx 默认配置中了解一些配置参数，当然最靠谱的途径还是看 Nginx 的文档。这里只简单尝试 Nginx，下面给出一个简单的配置：

```
server {
listen 80;                  # 服务器监听端口
server_name 110.110.110.110; # 这里写你的域名或者公网IP
charset      utf-8;         # 编码
client_max_body_size 75M;    # 之前写的关于GET和POST的区别，这里应该是原因之一吧

```

```
    uwsgi_param UWSGI_SCRIPT manage:app; （比如你测试用test.py文件，文件中app = Flask(__name__)，那
)           # 项目的主程序
    }
}
```

修改完成后我们将上述内容保存，稍作解释：server_name 可以是域名，也可以写 ip 地址，uwsgi_pass 是表明 Nginx 与 uwsgi 的交流方式，我这里用的是 sock 文件，当然你也可以用指定端口号的形式，具体可以看这里。
重启 Nginx：

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

这时刷新一下之前打开的服务器公网 ip（或绑定的域名），这时看到的就不是「Welcome to Nginx」，而是「502 Bad Way」，因为我们还没有启动 uWSGI，
编写uwsgi配置文件
在项目文件根目录新建配置文件uwsgi.ini（uwsgi支持多种配置文件格式, xml, ini, json等）
vim uwsgi.ini
写入配置内容如下：

```
[uwsgi]

socket = 127.0.0.1:8000             # uwsgi的监听端口

plugins = python                    # 这行一定要加上，不然请求时会出现-- unavailable modifier reques

chdir = /home/chihu                # 项目根目录

wsgi-file = manage.py               # flask程序的启动文件

callable = app                      # 程序变量名
```

输入:wq保存配置文件，可以通过uwsgi uwsgi.ini来启动uwsgi。

稍稍解释一下，socket 指定的是与 nginx 进行通信的端口文件。
上面的内容都是可以通过 uwsgi 命令的参数指定的，在命令行中敲入一行命令就可以了，为了「可持续发展」，当然是用文件保存下来比较好。
通过 uwsgi 命令，--ini 参数：

```
$ uwsgi --ini ini的文件名.ini &
```

指定配置文件，后台运行 uwsgi， 这时再刷新一下之前打开的页面，就可以看到应用正常运行了。
我尝试了在一台服务器上运行多个应用，其实只需要改一下文件名，分别处理 uWSGI 和 Nginx 的配置文件即可（Nginx 的配置，可以写在同一个文件中，写两个 server 就行了）
常用命令
nginx 常用命令
启动命令：

```
$ sudo nginx
```

或

```
$ sudo nginx -s stop
```

平滑启动 nginx

```
sudo nginx -s reload
```

所谓平滑启动就是在不停止 nginx 的情况下，重启 nginx，重新加载配置文件，用新的工作进程代替旧的工作进程。

STEP 4: 进程监控

原本打算用uwsgi emperor的方式运行uwsgi即让uwsgi成为守护进程(daemon)，但是折腾了一天还是没解决问题，遂换supervisor。（折腾uwsgi emperor到后面发现问题是，当单独使用uwsgi的时候一切正常，但一使用uwsgi emperor就会出现os.environ.get()获取不到环境变量的问题，但环境变量明明可以通过env看到，直接在Python终端上执行os.environ.get()也可以获取到.....等有时间再回过头了折腾。）

单独一个uwsgi程序运行短时间可能没问题，但是网络状况瞬息万变，万一uwsgi进程挂了网站也就访问不了了.....为了防止出现意外情况，还需要一个监控程序来监控uwsgi的状况，这里该supervisor出场了。具体什么是supervisor可以Google之。

1. 首先安装supervisor

```
sudo apt-get install supervisor
```

2. 生成默认配置文件

```
echo_supervisord_config > supervisord.conf
```

3. 加入监控程序的配置

```
[program:project_name]          # project_name这里写上你的项目名称，如我的为[project:chihu]
command = uwsgi --ini /home/chihu/uwsgi.ini  # 跟手动启动的命令一样
stopsignal=QUIT
autostart=true
autorestart=true
stdout_logfile=/var/log/uwsgi/supervisor_chihu.log      # 运行日志
stderr_logfile=/var/log/uwsgi/supervisor_chihu_err.log  # 错误日志
```

4. 启动supervisord

```
supervisorctl reload
```

5. 检查uwsgi进程是否正常运行

```
ps aux|grep uwsgi
```

如果一切正常，此时应该可以看到uwsgi进程

尝试kill掉uwsgi进程看supervisor会不会重新启动一个新的uwsgi进程

若再次通过ps aux|grep uwsgi查看发现有新的uwsgi进程在运行，那差不多可以祝你成功了, God bless you。

ERROR

supervisorctl reload

提示错误

error: <class 'socket.error'>, [Errno 2] No such file or directory: file: /usr/lib/python2.7/socket.py line: 224

issue

只需手动启动supervisord即可：

```
sudo supervisord -c configfile.conf
```

• 这里涉及到一个启动程序的用户的问题，好几次莫名不能通过os.environ.get()到系统中的环境变量，导致程序出错。一步一步排查猜测是因为启动程序的用户不匹配的问题，遂将启动程序的用户从root改为普通的登陆用户，问题解决，但这里还有疑问，因为环境变量是设置在/et/environment当中的，按理这里设置的应该是全局都可以访问的环境变量，但是当用root用户启动程序时就不能访问到环境变量，为了这个问题折腾了好几天，总算解决了，但是当中的疑惑还是需要好好思考探索。

后记：

• 折腾了两天终于搞定，有时候教程看着简单，你的配置也和教程一样，但就是由于各种错误无法运行，虽然说计算机非0即1，是我们人类最忠实可靠的伙伴，但是有时候就差那么一点点，有可能是软件环境不对亦或是我们电脑的打开方式不对...总之，就是要有那么个折腾的过程，所谓吃一堑长一智，不折腾折腾估计过后就忘了怎么回事了，还是要勤动手不放弃，虽然被一个uwsgi emperor折腾得快要怀疑人生了，但我还是坚定不移地...改道supervisor了：）

• ---

初次写教程，文中可能有疏漏或写得不够恰当的地方，还请各位看官多多包涵，欢迎指正和交流。如果部署的过程中有问题也欢迎留言，虽然不能保证可以解决。

-EOF-

参考：

Nginx+uwsgi+supervisor在Ubuntu上部署flask应用
http://www.cnblogs.com/dspace/archive/2016/07/06/5647587.html #

阿里云部署 Flask + WSGI + Nginx 详解
http://www.cnblogs.com/Ray-liang/p/4173923.html

写给新手看的Flask+uwsgi+Nginx+Ubuntu部署教程
http://www.cnblogs.com/knarfeh/p/5616515.html

https://segmentfault.com/a/1190000004294634 # uwsgi及Nginx配置
http://vladikk.com/2013/09/12/serving-flask-with-nginx-on-ubuntu/#comment-2401229330 # uwsgi emperor
http://letgoof.me/2013/deploy-django-project-with-uwsgi-nginx-and-supervisor/ # supervisor配置
http://liyangliang.me/posts/2015/06/using-supervisor/ # supervisor & supervisord
1 Why do I need nginx when I have uWSGI
2 Web开发技术发展历史
3 uWSGI 文档
4 Web Server Gateway Interface
5 PEP-3333
6 digitalocean tutorials

推荐阅读

一对"不知羞耻"的母女，撕开了我们致命的痛点，这片让人深思
阅读 11,769

Ab们再不努力，就要被杨超越吊打了
阅读 6,925

《中餐厅》黄晓明被骂的背后，藏着一个杠精诞生的底层逻辑
阅读 22,245

韩信为什么活埋了母亲?
阅读 21,199

开发部署提速8倍！这款IDE插件了解一下?
阅读 8,351