Django, Apache and mod_python on Ubuntu 8.04 (Hardy)

Updated Friday, June 1, 2018 by Linode

Written by Linode

Contribute on GitHub

Report an Issue (https://github.com/linode/docs/issues/new?title=Django%2c%20Apache%20and%20mod_python%20on%20Ubuntu%208.04%20%28Hardy%29%20Proposed%20Changes&body=Link%3A https%3A%2F%2Flinode.com%2fdocs%2fdevelopment%2fframeworks%2fdjango-apache-and-modpython-on-ubuntu-8-04-hardy%2f%0A%23%23%20Issue%0A%0A%23%23%20Suggested%20Fix%0A&labels=inaccurate guide) | View File (https://github.com/linode/docs/blob/master/docs/development/frameworks/django-apache-and-modpython-on-ubuntu-8-04-hardy/index.md) | Edit File (https://github.com/linode/docs/edit/develop/docs/development/frameworks/django-apache-and-modpython-on-ubuntu-8-04-hardy/index.md)

> **Deprecated**
> This guide has been deprecated and is no longer being maintained.

Django is a web development framework for the Python programing language. It enables rapid development, while favoring pragmatic and clean design. Django was initially developed for use in a newspaper's website division, and as a result the Django framework is very well suited to developing content-centric applications. It's also very flexible in its ability to facilitate many complex content management operations.

This guide provides an introduction to getting started with the Django framework. Although Ubuntu Hardy includes Django packages, these contain a dated version of the Django framework, in the 0.9x series. We've decided to install the most recent stable release of Django instead. This provides the best possible balance between the stability and support of the Ubuntu-Hardy release, and the most current Django API. In general the Apache, plus mod_python, plus Django is accepted as the idea setup for beginning Django deployments, although the framework is quite flexible with regards to how applications can be deployed. There are many base platforms that you may consider in the future as your needs grow and change.

We assume that you've completed the getting started guide (/docs/getting-started/) and have a running and up to date Ubuntu 8.04 (Hardy) system. Furthermore, you will want to have a running Apache web server (/docs/web-servers/apache/apache-2-web-server-on-ubuntu-8-04-lts-hardy/) and a functional MySQL database (/docs/databases/mysql/use-mysql-relational-databases-on-ubuntu-8-04-hardy/) installed.

Installing Python Dependencies

Make sure your package repositories and installed programs are up to date by issuing the following commands:

```
apt-get update
apt-get upgrade --show-upgraded
```

There are a number of packages that we need to install before we can deploy a Django application. The following command will download and install all of these dependencies:

```
apt-get install libapache2-mod-python python-mysqldb
```

This installs mod_python, which embeds a Python interpreter in the Apache web server (`libapache2-mod-python`), MySQL database bindings for Python, and version 1.02 of the Django web framework.

You may also want to install the following libraries and tools with apt:

- `python-setuptools` - Setup Tools (https://pypi.python.org/pypi/setuptools) provides a method for installing and managing Python packages/modules ("eggs"). This is helpful if you need to use a Python module that isn't available through `apt` .
- `python-psycopg2` - Provides an interface for using PostgreSQL in Python, if you prefer PostgreSQL to MySQL. You will also need to install the `postgresql` server. If you are running PostgreSQL you may not need to install or use MySQL and `python-mysqldb` .
- `python-sqlite` - Enables Python to access SQLite databases. These create a full-featured database inside a file. While this is often not suitable for high traffic production environments, it is often quite useful for testing and smaller deployments.

JSON (https://www.json.org/) and YAML

re are many additional Python-related packages in the operating system repositories. You can search the package database using the `apt-cache search python` command. If you need more information about a package, use the `apt-cache show [package-name]` command.

Downloading and Installing Django

Because we're not installing Django from the Ubuntu repository we must download the sources from the Django project's download page (https://www.djangoproject.com/download/). The current version is version 1.1 and you can download it using `wget` :

```
wget http://www.djangoproject.com/download/1.1/tarball/
```

After downloading the tarball, extract it using tar:

```
tar xzvf Django-1.1.tar.gz
```

Then move (`cd`) into the newly created `Django-1.1` directory (or later, if you've downloaded an alternate version) and use the following command to install Django:

```
python setup.py install
```

Be sure to run this with root privileges. The setup program will guide you through the remainder of the installation.

We recommend saving your source files and source tarballs for future reference. You might consider downloading and saving files in a `/src/` or `~/src` directory.

Configuring Apache

With all of the dependencies installed, we must configure Apache for virtual hosting. You will want to insert a `<Location >` block inside of the virtual hosting block for the domain where you want the Django application to run. The location block looks like this:

Apache Virtual Host Configuration

```
1  <Location "/">
2      SetHandler python-program
3      PythonHandler django.core.handlers.modpython
4      SetEnv DJANGO_SETTINGS_MODULE mysite.settings
5      PythonDebug Off
6  </Location>
```

You will need to change the `mysite.settings` to correspond to the settings file for your application in the Python path. The Python path is specific to the instance and version of Python that you're using and can be modified in your Python settings. If you want to store your Django application in another location, we'll need to specify a `PythonPath` variable in the `<Location >` block above by adding the following line:

Apache Virtual Host Configuration

```
1  PythonPath "['/srv/www/brackley.net/application'] + sys.path"
```

This line will allow mod_python to look for your settings file in the `/srv/www/brackley.net/application` directory, for an application in the "brackley.net" virtual host entry.

The `<Location >` block tells Apache what to do when a request comes in for a given URL location. For instance, if the above block is located in the VirtualHost entry for the `example.com` domain, then all requests for the URL `http://example.com/` would be directed to the Django application. In this case you do not need to set a `DocumentRoot` for this VirtualHost.

Hosting Static Content

If you wanted to have a static page located at the root of the domain and only use Django to power a blog at the URL `http://example.com/blog/` , the above block would begin with `<Location "/blog">` . In this situation, you would need to set up a DocumentRoot to contain the files for the static portion of the site.

Typically, Django applications use a secondary "media" web server to more efficiently serve static content like images, video, audio, and even static text resources. This permits more effective scaling possibilities. If you need to turn off Django and mod_python for a particular URL, add a second location block, like so:

Apache Virtual Host Configuration

In the above example, this would allow any static content requested with the URL `http://example.com/files/` to be served without Django interference. An alternate, and potentially easier solution, would use a second VirtualHost for all non-Python content.

Hosting Multiple Django Applications

The easiest way to host multiple Django applications with one instance of Apache is to place each application in its own virtual host. If, however, you need to host more than one application within a single VirtualHost entry you'll need specify different locations in `<Location >` blocks *within* that VirtualHost entry. Here are two example location blocks that would be inserted in your VirtualHost entry:

Apache Virtual Host Configuration

```
<Location "/lollipop">
    SetHandler python-program
    PythonHandler django.core.handlers.modpython
    SetEnv DJANGO_SETTINGS_MODULE lollipop.site.settings
    PythonDebug Off
    PythonInterpreter lollipop
</Location>

<Location "/funnyjoke">
    SetHandler python-program
    PythonHandler django.core.handlers.modpython
    SetEnv DJANGO_SETTINGS_MODULE funnyjoke.site.settings
    PythonDebug Off
    PythonInterpreter funnyjoke
</Location>
```

We'll note that the `PythonInterpreter` option needs to be set in these situations to avoid confusing mod_python.

Using Django

Once you have the base system installed and mod_python configured properly with Django, the majority of your time can be spent developing your application. There are, however, a few concerns of which you should be aware.

Because of the way that mod_python works, it's necessary to restart the web server whenever you update, change, or modify your Django application. This is because of the way that mod_python caches code. To restart Apache, issue the following command:

```
/etc/init.d/apache2 restart
```

As the site and your Django application begins receiving additional traffic, there are a number of steps you can take to scale your infrastructure with Django to increase performance. Some of these approaches are fairly simple and straightforward, while others may take much longer.

The first step is to separate services onto different servers. If you're having performance issues, move the database (e.g. MySQL or PostgreSQL) onto its own server or even a cluster of database servers. We alluded to this earlier with regard to static files, but it's often easier and more efficient to use a separate high-performance web server like nginx or lighttpd for static content. Such a server may also run on a separate Linode, isolated from the Apache instance running the Django application. Advanced solutions including front end reverse proxies like Squid, or hosting duplicate copies of your application servers and using a round-robin DNS setup, can offer you a great deal of scalability for high-demand situations.

More Information

You may wish to consult the following resources for additional information on this topic. While these are provided in the hope that they will be useful, please note that we cannot vouch for the accuracy or timeliness of externally hosted materials.

- The Django Project Home Page (https://www.djangoproject.com/)
- The Django Project Introductory Tutorial (https://docs.djangoproject.com/en/dev/intro/tutorial01/#intro-tutorial01)
- The Django Book (http://www.djangobook.com/)
- Deploying Django Applications (http://www.djangobook.com/en/2.0/chapter12/)
- Hello World" Django Application (http://runnable.com/UWRVp6lLuONCAABD/hello-world-in-django-for-python)

Join our Community

Find answers, ask questions, and help others. (https://www.linode.com/community/questions/)