TUTORIAL

How To Serve Django Applications with Apache and mod_wsgi on Ubuntu 14.04

Ubuntu Apache Python Django Python Frameworks

By Justin Ellingwood
Published on March 18, 2015 338.6k

Introduction

Django is a powerful web framework that can help you get your Python application or website off the ground quickly. Django includes a simplified development server for testing your code locally, but for anything even slightly production related, a more secure and powerful web server is required.

In this guide, we will demonstrate how to install and configure Django in a Python virtual environment. We'll then set up Apache in front of our application so that it can handle client requests directly before passing requests that require application logic to the Django app. We will do this using the `mod_wsgi` Apache module that can communicate with Django over the WSGI interface specification.

Prerequisites and Goals

In order to complete this guide, you should have a fresh Ubuntu 14.04 server instance with a non-root user with `sudo` privileges configured. You can learn how to set this up by running thorugh our initial server setup guide.

We will be installing Django within a Python virtual environment. Installing Django into an environment specific to your project will allow your projects and their requirements to be handled separately.

Once we have our application up and running, we will configure Apache to interface with the Django app. It will do this with the `mod_wsgi` Apache module, which can translate HTTP requests into a predictable application format defined by a specification called WSGI. You can find out more about WSGI by reading the linked section on this guide.

Let's get started.

Install Packages from the Ubuntu Repositories

To begin the process, we'll download and install all of the items we need from the Ubuntu repositories. This will include the Apache web server, the `mod_wsgi` module used to interface with our Django app, and `pip`, the Python package manager that can be used to download our Python-related tools.

If you are using Django with Python 2, the commands you need are:

```
n-pip apache2 libapache2-mod-wsgi
```

with Python 3, you will need an alternative Apache module. The ase are:

```
sudo apt-get update
sudo apt-get install python3-pip apache2 libapache2-mod-wsgi-py3
```

When operating *outside* of a virtual environment for the remainder of the tutorial, if you are using Python 3, replace `pip` with `pip3`.

Configure a Python Virtual Environment

Now that we have the components from the Ubuntu repositories, we can start working on our Django project. The first step is to create a Python virtual environment so that our Django project will be separate from the system's tools and any other Python projects we may be working on.

We need to install the `virtualenv` command to create these environments. We can get this using `pip`:

```
sudo pip install virtualenv
```

With `virtualenv` installed, we can start forming our project. Create a directory where you wish to keep your project and move into the directory:

```
mkdir ~/myproject
cd ~/myproject
```

Within the project directory, create a Python virtual environment by typing:

```
virtualenv myprojectenv
```

This will create a directory called `myprojectenv` within your `myproject` directory. Inside, it will install a local version of Python and a local version of `pip`. We can use this to install and configure an isolated Python environment for our project.

Before we install our project's Python requirements, we need to activate the virtual environment. You can do that by typing:

```
source myprojectenv/bin/activate
```

Your prompt should change to indicate that you are now operating within a Python virtual environment. It will look something like this: `(myprojectenv)user@host:~/myproject$`.

With your virtual environment active, install Django with the local instance of `pip` by typing:

```
pip install django
```

Create and Configure a New Django Project

Now that Django is installed in our virtual environment, we can create the actual Django project files.

Create the Django Project

Since we already have a project directory, we will tell Django to install the files here. It will create a second level directory with the actual code, which is normal, and place a management script in this directory. The key to this is the dot at the end that tells Django to create the files in the current

Adjust the Project Settings

th our newly created project files is adjust the settings. Open the
or:

```
/
```

We are going to be using the default SQLite database in this guide for simplicity's sake, so we don't actually need to change too much. We will focus on configuring the static files directory, where Django will place static files so that the web server can serve these easily.

At the bottom of the file, we will add a line to configure this directory. Django uses the `STATIC_ROOT` setting to determine the directory where these files should go. We'll use a bit of Python to tell it to use a directory called "static" in our project's main directory:

```
STATIC_ROOT = os.path.join(BASE_DIR, "static/")
```

Save and close the file when you are finished.

Complete Initial Project Setup

Now, we can migrate the initial database schema to our SQLite database using the management script:

```
cd ~/myproject
./manage.py makemigrations
./manage.py migrate
```

Create an administrative user for the project by typing:

```
./manage.py createsuperuser
```

You will have to select a username, provide an email address, and choose and confirm a password.

We can collect all of the static content into the directory location we configured by typing:

```
./manage.py collectstatic
```

You will have to confirm the operation. The static files will be placed in a directory called `static` within your project directory.

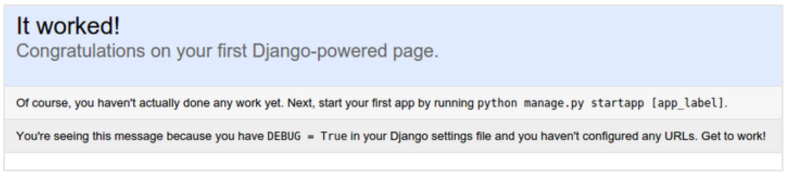
Finally, you can test your project by starting up the Django development server with this command:

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

In your web browser, visit your server's domain name or IP address followed by `:8000`:

```
http://server_domain_or_IP:8000
```

You should see the default Django index page:

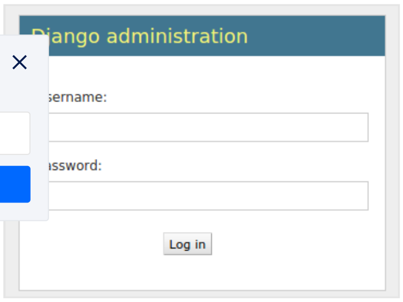


It worked!
Congratulations on your first Django-powered page.

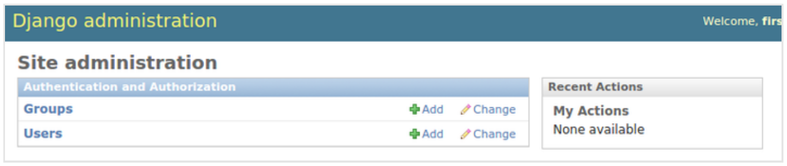
Of course, you haven't actually done any work yet. Next, start your first app by running `python manage.py startapp [app_label]`.

You're seeing this message because you have `DEBUG = True` in your Django settings file and you haven't configured any URLs. Get to work!

If you append `/admin` to the end of the URL in the address bar, you will be prompted for the administrative username and password you created with the `createsuperuser` command:



After authenticating, you can access the default Django admin interface:



When you are finished exploring, hit CTRL-C in the terminal window to shut down the development server.

We're now done with Django for the time being, so we can back out of our virtual environment by typing:

```
deactivate
```

Configure Apache

Now that your Django project is working, we can configure Apache as a front end. Client connections that it receives will be translated into the WSGI format that the Django application expects using the `mod_wsgi` module. This should have been automatically enabled upon installation earlier.

To configure the WSGI pass, we'll need to edit the default virtual host file:

```
sudo nano /etc/apache2/sites-available/000-default.conf
```

We can keep the directives that are already present in the file. We just need to add some additional items.

To start, let's configure the static files. We will use an alias to tell Apache to map any requests starting with `/static` to the "static" directory within our project folder. We collected the static assets there earlier. We will set up the alias and then grant access to the directory in question with a directory block:

```
<VirtualHost *:80>
    . . .

    Alias /static /home/user/myproject/static
    <Directory /home/user/myproject/static>
        Require all granted
    </Directory>

</VirtualHost>
```

Next, we'll grant access to the `wsgi.py` file within the second level project directory where the Django code is stored. To do this, we'll use a directory section with a file section inside. We will grant access to the file inside of this nested construct:

```
<VirtualHost *:80>
    . . .
    <Directory /home/user/myproject/myproject>
```

```
anted
```

After this is configured, we are ready to construct the portion of the file that actually handles the WSGI pass. We'll use daemon mode to run the WSGI process, which is the recommended configuration. We can use the `WSGIDaemonProcess` directive to set this up.

This directive takes an arbitrary name for the process. We'll use `myproject` to stay consistent. Afterwards, we set up the Python path to the project's parent directory. This will be `/home/user/myproject` in this guide. Since we used a virtual environment, we will also need to set the Python home to the root of our virtual environment. This way, Apache can find all of the other Python code needed to run our project.

Afterwards, we need to specify the process group. This should point to the same name we selected for the `WSGIDaemonProcess` directive (`myproject` in our case). Finally, we need to set the script alias so that Apache will pass requests for the root domain to the `wsgi.py` file:

```
<VirtualHost *:80>
    . . .

    Alias /static /home/user/myproject/static
    <Directory /home/user/myproject/static>
        Require all granted
    </Directory>

    <Directory /home/user/myproject/myproject>
        <Files wsgi.py>
            Require all granted
        </Files>
    </Directory>

    WSGIDaemonProcess myproject python-path=/home/user/myproject python-home=/home/use
    WSGIProcessGroup myproject
    WSGIScriptAlias / /home/user/myproject/myproject/wsgi.py

</VirtualHost>
```

When you are finished making these changes, save and close the file.

Wrapping Up Some Permissions Issues

If you are using the SQLite database, which is the default used in this article, you need to allow the Apache process access to this file.

To do so, the first step is to change the permissions so that the group owner of the database can read and write. The database file is called `db.sqlite3` by default and it should be located in your base project directory:

```
chmod 664 ~/myproject/db.sqlite3
```

Afterwards, we need to give the group Apache runs under, the `www-data` group, group ownership of the file:

```
sudo chown :www-data ~/myproject/db.sqlite3
```

In order to write to the file, we also need to give the Apache group ownership over the database's parent directory:

```
sudo chown :www-data ~/myproject
```

Once these steps are done, you are ready to restart the Apache service to implement the changes

You should now be able to access your Django site by going to your server's domain name or IP

rt. The regular site and the admin interface should function as

ngo project in its own virtual environment. We've configured Apache with `mod_wsgi` to handle client requests and interface with the Django app.

Django makes creating projects and applications simple by providing many of the common pieces, allowing you to focus on the unique elements. By leveraging the general tool chain described in this article, you can easily serve the applications you create from a single server.

About the authors

Justin Ellingwood
Senior Technical Writer @DigitalOcean

Still looking for an answer?

How To Perform CRUD Operations in MongoDB Using PyMongo on Ubuntu 20.04
Tutorial

Jumping Into Django Models
Tutorial

Comments

44 Comments

Leave a comment...

Sign In to Comment

xRahul June 18, 2015

I wasn't able to set my virtual host using the above method. after setting my virtual host, it showed internal server error 500 on even other sites I had.
Can you clarify why that occured?!

Reply Report

jellingwood June 18, 2015

@rahulgr8888: This tutorial's prerequisites assume that you are operating on a fresh Ubuntu server without other sites configured. If you can post some of your logs, that would give us a better place to start troubleshooting.

Reply Report

PepSeeker July 1, 2015

Hi, i have the same error even if it is my only site. Here is my Apache log :
[Wed Jul 01 13:13:59.196803 2015] [:error] [pid 6488] [remote 127.0.0.1:9196] Traceback (most recent call last):
[Wed Jul 01 13:13:59.196850 2015] [:error] [pid 6488] [remote 127.0.0.1:9196] File "/var/www/GestionStock/GestionStock/wsgi.py", line 18, in <module>
[Wed Jul 01 13:13:59.196856 2015] [:error] [pid 6488] [remote 127.0.0.1:9196] from django.core.wsgi import get*wsgi*application
[Wed Jul 01 13:13:59.196879 2015] [:error] [pid 6488] [remote 127.0.0.1:9196] ImportError: No module named 'django'

Thank you to answer.

Reply Report

jellingwood July 1, 2015

@PepSeeker: It looks like it isn't finding your Django installation. This could be because the `python-path` specified through the `WSGIDaemonProcess` directive in your virtual host file isn't pointing to the place where your Django files are located. You should check your virtual environment directory to see if you can find Django in there. Also, you seem to be running this out of `/var/www`, which isn't really following the procedure laid out here. You could end up running into additional permission issues and your components might not be where these instructions expect them to be. I think I would probably have difficulty troubleshooting this much more with this setup.

Reply Report

PepSeeker July 2, 2015

@jellingwood Thank you for your reply, I finally figured out why this wasn't working. The python path wasn't the fault. I found on a forum that i need to activate the virtual environment in the wsgi.py file. So I added :

```
activate_this = '/var/www/GestionStock/GestionStockenv/bin/activate_this.py'
exec(compile(open(activate_this,"rb").read(),activate_this, 'exec'), dict(__file_
```

and this did the trick.

Reply Report

ryan81s June 24, 2015

I keep getting a "permission denied" with WSGIScriptAlias, even though the "document root" and related directives are configured correctly. The application files are set to "rwx" where the apache user has group access through "www-data". The site is symlinked through "/var/www/html/project_name". I've double and triple checked the virtual environment and directory structure, and the permissions. Not sure what the issue is...

```
    DocumentRoot /var/www/html

    Alias /static /piv/static
    <Directory /piv/piv>
                    i.py>
                    quire all granted


                    piv python-path=/piv/piv:/lib/python2.7/site-packages
                    iv
                    iv /piv/piv/wsgi.py
```

Reply Report

0 **rajpar** July 7, 2015
I follow your article and everything went fine but apache prompts following whenever i type IP of my system. Forbidden You don't have permission to access / on this server.
Reply Report

0 **mmhawker** July 7, 2015
Do you also happen to have this error message in console ?
"`AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 127.0.1.1. Set the 'ServerName' directive globally to suppress this message

Reply Report

0 **rajpar** July 8, 2015
[deleted]
Reply Report

0 **rajpar** July 8, 2015
[deleted]
Reply Report

0 **rajpar** July 8, 2015
Thanks to reply.
no, i don't face such kind of message at console. I also tries loopback IP (127.0.1.1) but it replies same message 'Forbidden, You don't have permission to access / on this server'.
Reply Report

0 **mmhawker** July 8, 2015
I got rid of "You don't have permission to access" this by double/triple/quadruple checking the correct paths
Reply Report

0 **rajpar** July 9, 2015
Don't get rid off me, i got exact same error as you mentioned in your first reply.
Kindly get me out off it.
Reply Report

0 **JZA** October 2, 2015
Make sure to revisit the logs, sometimes the system python gets executed and causes havok on the wsgi.py which prompts to an early stop. Checking ALL the error logs will show if mod_python somehow mess things up.
Reply Report

2 **abhishekb** September 10, 2015
you missed a step probably: `chown www-data. .`
Reply Report

1 **gustavobelduma** April 15, 2016
Esta linea no esta en el tutorial y es de gran importancia para que se ejecute con normalidad el proyecto
Reply Report

0 **alinajafi** May 13, 2017
It's better to type `chown www-data .`
Reply Report

0 **JZA** October 1, 2015
I didn't see any site activation. Did I miss something here?

Reply Report

...u're using python 3 with django you have to install libapache2-mod-wsgi-
...or the past two weeks because WSGI was using the wrong version of

...ember 16, 2015
0
@caldwellysr: I'm sorry you had so much trouble with the article as written. Thank you though for sharing back the root cause and solution you found. I've updated the article so that future readers won't run into the same issue. Thanks again for helping out!

Reply Report

0
dms November 30, 2015
Great tutorial! I'm not sure about Ubuntu, but under CentOS, *pip install mod_wsgi* automatically uses the proper version of python but it doesn't configure apache to use it. I've been getting "Invalid command WSGIDaemonProcess" errors because apache isn't loading the mod_wsgi module. My fix was to create a symbolic link named "mod_wsgi.so" from apache's modules directory to the .so file in the python3/django virtual environment, and create a config file included by httpd.conf that contains: "LoadModule wsgi_module modules/mod_wsgi.so".

Reply Report

0
caldwellysr November 30, 2015
I should have clarified… This is a great article and with it I have gotten farther with my first django installation than with any other tutorial I've been able to find. So thank you!

@dms I was using Ubuntu and it once I figured out that I was using the wrong version of mod_wsgi I didn't have any issues.

Reply Report

1
koenigcochran November 30, 2015
I apologize for being late to the party. I appreciate your tutorial–it's precisely what I need. I've followed your tutorial to the point that I can test Django's toy-server. However, after executing "./manage.py runserver 0.0.0.0:8000" I cannot access the webpage from my browser at "http://IP-ADDRESS:8000." The browser complains that the connection times out. That said, "http://IP-ADDRESS" quickly loads the default Apache Ubuntu Index page. Any help would be much appreciated!

PS I am not using a virtual environment. The permissions on my server have twisted my arm into using sudo for many commands that you do not. When using sudo, I noticed I would have to use the absolute path for all executables and scripts–even after activating the venv, which was too much of a pain for this quick exercise. Thanks so much!

Reply Report

0
cutecode December 14, 2015
Great technical article!
But I encounter **Forbidden You don't have permission to access / on this server** problem when I deployed with root user.

A great answer from StackOverflow.
http://stackoverflow.com/a/14623574.

Reply Report

0
netfusion January 13, 2016
I had the same permissions issue that many others had - apache could not access something outside of the /var/www/ directory. The last thing that I wanted to do though was to start changing permissions for all directories using chmod.

My mistake was simple though - in the CONFIGURE APACHE step in the article I did not substitute my default ec2 user "ubuntu" for the word "user" everywhere in the .conf file.

For example: I had to use "WSGIScriptAlias / /home/ubuntu/myproject/myproject/wsgi.py" as opposed to "WSGIScriptAlias / /home/user/myproject/myproject/wsgi.py", and the same for all other lines. Simple, yes, but I still missed it for a while. All works fine now.

Thanks as well for the really helpful article. It came in handy after spending all day trying to get Django working on Amazon Linux AMI and then Amazon Elastic Beanstock without effect.

Reply Report

0
gajjar8055 May 9, 2016

Reply Report

16

```
'user/myproject')
```

in the `wsgi.py`.

Reply Report

0
adeyoga October 3, 2016
guys i need help,

i wanst able to see my django web using this method , i dont know why. Im using this method extremly the same way as this tutorial says and "500 internal server error" is the result. I really need help. Anyone knows why?

Reply Report

0
addohm October 9, 2016
I struggled here:

```
WSGIScriptAlias / /home/user/myproject/myproject/wsgi.py
```

My understanding of your article is that the FIRST myproject folder in the path is where you ran the command `django-admin startproject myproject`

If that's true then the path of your wsgi.pi would be:

```
/home/user/myproject/myproject/myproject/wsgi.py
```

Reply Report

0
jellingwood October 10, 2016
@addohm: Sorry to hear you had trouble. You *may* have an extra `myproject` directory if you left off the period at the end of this command:

```
django-admin.py startproject myproject .
```

Without the period, Django will create another directory and place the generated files within. If you include the period, however, Django will assume that you are already in the working directory you want to use and will dump the files in the current directory.

Since we manually created our directory to set up `virtualenv`, this guide uses the period syntax to avoid the extra directory structure. Hope that helps.

Reply Report

0
songyang October 17, 2016
Hi, this is really a great article.

And after following all the steps above, i have successfully deployed the demo project on Apache.

But i have trouble when deployed my own project on Apache. After doing all the steps above with some adjustment, in the browser, when i type 'localhost', it returns 500 ERROR. And i check my Apache error log, it says "ImportError: No module named <MY-APP-NAME>.apps" and "RuntimeError: populate() isn't reentrant".

My project folder is at '/var/www/<MY-PROJECT-NAME>'

```
<MY-PROJECT-NAME>
    -<MY-PROJECT-NAME>
        -__init__.py
        -settings.py
        -urls.py
        -wsgi.py
    -<MY-APP-NAME>
        -migrations/
        -static/
        -models.py
```

```
LDER>
```

o...

And I don't know why it produce such an error... I can run the django app using: 'python manage.py runserver'. But i just can't deploy it on Apache.

Any advice will be appreciated. Thanks.

Reply Report

0

avezaatb November 18, 2016

Very nice tutorial, the only feedback I have is that the change over from python(2) to python3 I struggled for a while to get the correct virtualenv set up. After some Googling I did find an easy solution that might be handy to add to this tutorial. To create a python3 virtual environment after installing for me only worked with this command 'virtualenv -p python3 [envname]'

Reply Report

0

w3resource2011 January 12, 2017

I have a similar kind of situation. Now what I want to do is, that my users should be able to access the app using mydomainname.com/foldername instead of ip:port. How to do that?

Reply Report

0

gustavobelduma March 6, 2017

Please correct the permit:

sudo chown :www-data ~/myproject/db.sqlite3
sudo chown :www-data ~/myproject

for:

sudo chown www-data ~/myproject/db.sqlite3
sudo chown www-data ~/myproject

If it is in Debian

Reply Report

0

cknolla April 15, 2017

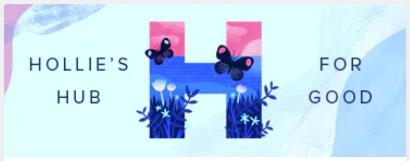
Another expertly written and empowering guide. Thanks Justin.

Reply Report

Load More Comments

GET OUR BIWEEKLY NEWSLETTER

Sign up for Infrastructure as a Newsletter.



HOLLIE'S HUB FOR GOOD

Working on improving health and education, reducing inequality, and spurring economic growth? We'd like to help.

BECOME A CONTRIBUTOR

You get paid; we donate to tech nonprofits.