



By: Justin Ellingwood

Subscribe



## How To Set Up Django with Postgres, Nginx, and Gunicorn on Ubuntu 16.04

57

Posted May 18, 2016 109.5k [POSTGRES](#) [DJANGO](#) [NGINX](#) [PYTHON FRAMEWORKS](#) [PYTHON](#) [UBUNTU](#) [UBUNTU 16.04](#)

### Introduction

Django is a powerful web framework that can help you get your Python application or website off the ground. 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 some components on Ubuntu 16.04 to support and serve Django applications. We will be setting up a PostgreSQL database instead of using the default SQLite database. We will configure the Gunicorn application server to interface with our applications. We will then set up Nginx to reverse proxy to Gunicorn, giving us access to its security and performance features to serve our apps.

### Prerequisites and Goals

In order to complete this guide, you should have a fresh Ubuntu 16.04 server instance with a non-root user with `sudo` privileges configured. You can learn how to set this up by running through our [initial server setup guide](#).

We will be installing Django within a 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 database and application up and running, we will install and configure the Gunicorn application server. This will serve as an interface to our application, translating client requests in HTTP to Python calls that our application can

process. We will then set up Nginx in front of Gunicorn to take advantage of its high performance connection handling mechanisms and its easy-to-implement security features.

Let's get started.

## Install the Packages from the Ubuntu Repositories

To begin the process, we'll download and install all of the items we need from the Ubuntu repositories. We will use the Python package manager `pip` to install additional components a bit later.

We need to update the local `apt` package index and then download and install the packages. The packages we install depend on which version of Python your project will use.

If you are using **Python 2**, type:

```
$ sudo apt-get update
$ sudo apt-get install python-pip python-dev libpq-dev postgresql postgresql-contrib nginx
```

If you are using Django with **Python 3**, type:

```
$ sudo apt-get update
$ sudo apt-get install python3-pip python3-dev libpq-dev postgresql postgresql-contrib nginx
```

This will install `pip`, the Python development files needed to build Gunicorn later, the Postgres database system and the libraries needed to interact with it, and the Nginx web server.

## Create the PostgreSQL Database and User

We're going to jump right in and create a database and database user for our Django application.

By default, Postgres uses an authentication scheme called "peer authentication" for local connections. Basically, this means that if the user's operating system username matches a valid Postgres username, that user can login with no further authentication.

During the Postgres installation, an operating system user named `postgres` was created to correspond to the `postgres` PostgreSQL administrative user. We need to use this user to perform administrative tasks. We can use `sudo` and pass in the username with the `-u` option.

Log into an interactive Postgres session by typing:

```
$ sudo -u postgres psql
```

You will be given a PostgreSQL prompt where we can set up our requirements.

First, create a database for your project:

```
postgres=# CREATE DATABASE myproject;
```

Every Postgres statement must end with a semi-colon, so make sure that your command ends with one if you are experiencing issues.

Next, create a database user for our project. Make sure to select a secure password:

```
postgres=# CREATE USER myprojectuser WITH PASSWORD 'password';
```

Afterwards, we'll modify a few of the connection parameters for the user we just created. This will speed up database operations so that the correct values do not have to be queried and set each time a connection is established.

We are setting the default encoding to UTF-8, which Django expects. We are also setting the default transaction isolation scheme to "read committed", which blocks reads from uncommitted transactions. Lastly, we are setting the timezone. By default, our Django projects will be set to use UTC. These are all recommendations from the Django project itself:

```
postgres=# ALTER ROLE myprojectuser SET client_encoding TO 'utf8';
postgres=# ALTER ROLE myprojectuser SET default_transaction_isolation TO 'read committed';
postgres=# ALTER ROLE myprojectuser SET timezone TO 'UTC';
```

Now, we can give our new user access to administer our new database:

```
postgres=# GRANT ALL PRIVILEGES ON DATABASE myproject TO myprojectuser;
```

When you are finished, exit out of the PostgreSQL prompt by typing:

```
postgres=# \q
```

## Create a Python Virtual Environment for your Project

Now that we have our database, we can begin getting the rest of our project requirements ready. We will be installing our Python requirements within a virtual environment for easier management.

To do this, we first need access to the `virtualenv` command. We can install this with `pip`.

If you are using **Python 2**, upgrade `pip` and install the package by typing:

```
$ sudo -H pip install --upgrade pip
$ sudo -H pip install virtualenv
```

If you are using **Python 3**, upgrade `pip` and install the package by typing:

```
$ sudo -H pip3 install --upgrade pip
$ sudo -H pip3 install virtualenv
```

With `virtualenv` installed, we can start forming our project. Create and move into a directory where we can keep our project files:

```
$ 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, Gunicorn, and the `psycopg2` PostgreSQL adaptor with the local instance of `pip`:

#### Note

Regardless of which version of Python you are using, when the virtual environment is activated, you should use the `pip` command (not `pip3`).

```
(myprojectenv) $ pip install django gunicorn psycopg2
```

You should now have all of the software needed to start a Django project.

## Create and Configure a New Django Project

With our Python components installed, 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 that we are defining the directory explicitly instead of allowing Django to make decisions relative to our current directory:

```
(myprojectenv) $ django-admin.py startproject myproject ~/myproject
```

At this point, your project directory (`~/myproject` in our case) should have the following content:

- `~/myproject/manage.py`: A Django project management script.
- `~/myproject/myproject/`: The Django project package. This should contain the `__init__.py`, `settings.py`, `urls.py`, and `wsgi.py` files.
- `~/myproject/myprojectenv/`: The virtual environment directory we created earlier.

### Adjust the Project Settings

The first thing we should do with our newly created project files is adjust the settings. Open the settings file in your text

The next thing we should do with our newly created project is to adjust the settings. Open the settings file in your text editor:

```
(myprojectenv) $ nano ~/myproject/myproject/settings.py
```

Start by locating the `ALLOWED_HOSTS` directive. This defines a list of the server's addresses or domain names that may be used to connect to the Django instance. Any incoming requests with a **Host** header that is not in this list will raise an exception. Django requires that you set this to prevent a certain class of security vulnerability.

In the square brackets, list the IP addresses or domain names that are associated with your Django server. Each item should be listed in quotations with entries separated by a comma. If you wish requests for an entire domain and any subdomains, prepend a period to the beginning of the entry. In the snippet below, there are a few commented out examples used to demonstrate:

~/myproject/myproject/settings.py

```
...
# The simplest case: just add the domain name(s) and IP addresses of your Django server
# ALLOWED_HOSTS = ['example.com', '203.0.113.5']
# To respond to 'example.com' and any subdomains, start the domain with a dot
# ALLOWED_HOSTS = ['.example.com', '203.0.113.5']
ALLOWED_HOSTS = ['your_server_domain_or_IP', 'second_domain_or_IP', ...]
```

Next, find the section that configures database access. It will start with `DATABASES`. The configuration in the file is for a SQLite database. We already created a PostgreSQL database for our project, so we need to adjust the settings.

Change the settings with your PostgreSQL database information. We tell Django to use the `psycopg2` adaptor we installed with `pip`. We need to give the database name, the database username, the database user's password, and then specify that the database is located on the local computer. You can leave the `PORT` setting as an empty string:

~/myproject/myproject/settings.py

```
...
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': 'myproject',
        'USER': 'myprojectuser',
        'PASSWORD': 'password',
        'HOST': 'localhost',
        'PORT': '',
    }
}
...
```

Next, move down to the bottom of the file and add a setting indicating where the static files should be placed. This is necessary so that Nginx can handle requests for these items. The following line tells Django to place them in a directory called `static` in the base project directory:

~/myproject/myproject/settings.py

```
...  
STATIC_URL = '/static/'  
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 PostgreSQL database using the management script:

```
(myprojectenv) $ ~/myproject/manage.py makemigrations  
(myprojectenv) $ ~/myproject/manage.py migrate
```

Create an administrative user for the project by typing:

```
(myprojectenv) $ ~/myproject/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:

```
(myprojectenv) $ ~/myproject/manage.py collectstatic
```

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

If you followed the initial server setup guide, you should have a UFW firewall protecting your server. In order to test the development server, we'll have to allow access to the port we'll be using.

Create an exception for port 8000 by typing:

```
(myprojectenv) $ sudo ufw allow 8000
```

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

```
(myprojectenv) $ ~/myproject/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:

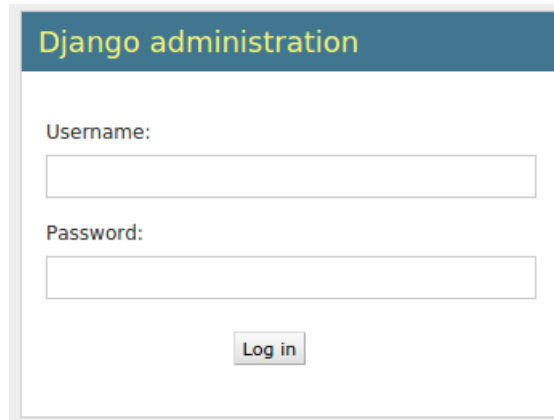
**It worked!**

Congratulations on your first Django-powered page.

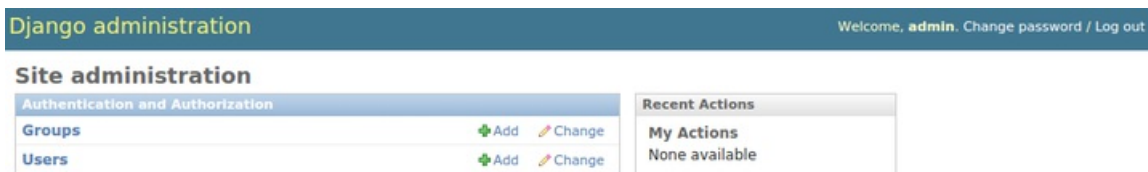
Of course, you haven't actually done any work yet. Next, start your first app by running `python manage.py startapp [app_label]`.

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:

A screenshot of the Django administration login page. It has a blue header with the text "Django administration". Below the header, there are two input fields: "Username:" and "Password:". At the bottom, there is a "Log in" button.

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

A screenshot of the Django administration interface. The header is blue with "Django administration" on the left and "Welcome, admin. Change password / Log out" on the right. Below the header, there is a "Site administration" section. It has a table with two rows: "Groups" and "Users". Each row has "Add" and "Change" links. To the right of the table, there is a "Recent Actions" section with a "My Actions" link and the text "None available".

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

## Testing Gunicorn's Ability to Serve the Project

The last thing we want to do before leaving our virtual environment is test Gunicorn to make sure that it can serve the application. We can do this by entering our project directory and using `gunicorn` to load the project's WSGI module:

```
(myprojectenv) $ cd ~/myproject
(myprojectenv) $ gunicorn --bind 0.0.0.0:8000 myproject.wsgi
```

This will start Gunicorn on the same interface that the Django development server was running on. You can go back and test the app again.

**Note:** The admin interface will not have any of the styling applied since Gunicorn does not know about the static CSS content responsible for this.

We passed Gunicorn a module by specifying the relative directory path to Django's `wsgi.py` file, which is the entry point to our application, using Python's module syntax. Inside of this file, a function called `application` is defined, which is used to communicate with the application. To learn more about the WSGI specification, click [here](#).

When you are finished testing, hit **CTRL-C** in the terminal window to stop Gunicorn.

We're now finished configuring our Django application. We can back out of our virtual environment by typing:

```
(myprojectenv) $ deactivate
```

The virtual environment indicator in your prompt will be removed.

## Create a Gunicorn systemd Service File

We have tested that Gunicorn can interact with our Django application, but we should implement a more robust way of starting and stopping the application server. To accomplish this, we'll make a systemd service file.

Create and open a systemd service file for Gunicorn with `sudo` privileges in your text editor:

```
$ sudo nano /etc/systemd/system/gunicorn.service
```

Start with the `[Unit]` section, which is used to specify metadata and dependencies. We'll put a description of our service here and tell the init system to only start this after the networking target has been reached:

```
/etc/systemd/system/gunicorn.service
```

```
[Unit]
Description=gunicorn daemon
After=network.target
```

Next, we'll open up the `[Service]` section. We'll specify the user and group that we want to process to run under. We will give our regular user account ownership of the process since it owns all of the relevant files. We'll give group ownership to the `www-data` group so that Nginx can communicate easily with Gunicorn.

We'll then map out the working directory and specify the command to use to start the service. In this case, we'll have to specify the full path to the Gunicorn executable, which is installed within our virtual environment. We will bind it to a Unix socket within the project directory since Nginx is installed on the same computer. This is safer and faster than using a network port. We can also specify any optional Gunicorn tweaks here. For example, we specified 3 worker processes in this case:

```
/etc/systemd/system/gunicorn.service
```

```
[Unit]
Description=gunicorn daemon
After=network.target

[Service]
User=sammy
Group=www-data
WorkingDirectory=/home/sammy/myproject
ExecStart=/home/sammy/myproject/myprojectenv/bin/gunicorn --access-logfile - --workers 3 --bind unix:/home/sammy/myproject/myproject.sock myproject.wsgi:application
```

Finally, we'll add an `[Install]` section. This will tell systemd what to link this service to if we enable it to start at boot. We want this service to start when the regular multi-user system is up and running:

```
/etc/systemd/system/gunicorn.service
```

```
[Unit]
Description=gunicorn daemon
After=network.target
```

```
[Service]
```



```
[Service]
User=sammy
Group=www-data
WorkingDirectory=/home/sammy/myproject
ExecStart=/home/sammy/myproject/myprojectenv/bin/gunicorn --access-logfile - --workers 3 --bind unix:/home/sammy/myproject/myproject.sock
myproject.wsgi:application

[Install]
WantedBy=multi-user.target
```

With that, our systemd service file is complete. Save and close it now.

We can now start the Gunicorn service we created and enable it so that it starts at boot:

```
$ sudo systemctl start gunicorn
$ sudo systemctl enable gunicorn
```

We can confirm that the operation was successful by checking for the socket file.

## Check for the Gunicorn Socket File

Check the status of the process to find out whether it was able to start:

```
$ sudo systemctl status gunicorn
```

Next, check for the existence of the `myproject.sock` file within your project directory:

```
$ ls /home/sammy/myproject
```

Output

```
manage.py myproject myprojectenv myproject.sock static
```

If the `systemctl status` command indicated that an error occurred or if you do not find the `myproject.sock` file in the directory, it's an indication that Gunicorn was not able to start correctly. Check the Gunicorn process logs by typing:

```
$ sudo journalctl -u gunicorn
```

Take a look at the messages in the logs to find out where Gunicorn ran into problems. There are many reasons that you may have run into problems, but often, if Gunicorn was unable to create the socket file, it is for one of these reasons:

- The project files are owned by the `root` user instead of a `sudo` user
- The `WorkingDirectory` path within the `/etc/systemd/system/gunicorn.service` file does not point to the project directory
- The configuration options given to the `gunicorn` process in the `ExecStart` directive are not correct. Check the following items:
  - The path to the `gunicorn` binary points to the actual location of the binary within the virtual environment
  - The `--bind` directive defines a file to create within a directory that Gunicorn can access
  - The `myproject.wsgi:application` is an accurate path to the WSGI callable. This means that when you're in the `WorkingDirectory`, you should be able to reach the callable named `application` by looking in the `myproject.wsgi` module (which translates to a file called `./myproject/wsgi.py`)

If you make changes to the `/etc/systemd/system/gunicorn.service` file, reload the daemon to reread the service definition and restart the Gunicorn process by typing:

```
$ sudo systemctl daemon-reload
$ sudo systemctl restart gunicorn
```

Make sure you troubleshoot any of the above issues before continuing.

## Configure Nginx to Proxy Pass to Gunicorn

Now that Gunicorn is set up, we need to configure Nginx to pass traffic to the process.

Start by creating and opening a new server block in Nginx's `sites-available` directory:

```
$ sudo nano /etc/nginx/sites-available/myproject
```

Inside, open up a new server block. We will start by specifying that this block should listen on the normal port 80 and that it should respond to our server's domain name or IP address:

`/etc/nginx/sites-available/myproject`

```
server {
    listen 80;
    server_name server_domain_or_IP;
}
```

Next, we will tell Nginx to ignore any problems with finding a favicon. We will also tell it where to find the static assets that we collected in our `~/myproject/static` directory. All of these files have a standard URI prefix of `/static`, so we can create a location block to match those requests:

`/etc/nginx/sites-available/myproject`

```
server {
    listen 80;
    server_name server_domain_or_IP;

    location = /favicon.ico { access_log off; log_not_found off; }
    location /static/ {
        root /home/sammy/myproject;
    }
}
```

Finally, we'll create a `location / {}` block to match all other requests. Inside of this location, we'll include the standard `proxy_params` file included with the Nginx installation and then we will pass the traffic to the socket that our Gunicorn process created:

`/etc/nginx/sites-available/myproject`

```
server {
    listen 80;
    server_name server_domain_or_IP;
```

```

location = /favicon.ico { access_log off; log_not_found off; }
location /static/ {
    root /home/sammy/myproject;
}

location / {
    include proxy_params;
    proxy_pass http://unix:/home/sammy/myproject/myproject.sock;
}
}

```

Save and close the file when you are finished. Now, we can enable the file by linking it to the `sites-enabled` directory:

```
$ sudo ln -s /etc/nginx/sites-available/myproject /etc/nginx/sites-enabled
```

Test your Nginx configuration for syntax errors by typing:

```
$ sudo nginx -t
```

If no errors are reported, go ahead and restart Nginx by typing:

```
$ sudo systemctl restart nginx
```

Finally, we need to open up our firewall to normal traffic on port 80. Since we no longer need access to the development server, we can remove the rule to open port 8000 as well:

```
$ sudo ufw delete allow 8000
$ sudo ufw allow 'Nginx Full'
```

You should now be able to go to your server's domain or IP address to view your application.

#### Note

After configuring Nginx, the next step should be securing traffic to the server using SSL/TLS. This is important because without it, all information, including passwords are sent over the network in plain text.

If you have a domain name, the easiest way get an SSL certificate to secure your traffic is using Let's Encrypt. Follow [this guide](#) to set up Let's Encrypt with Nginx on Ubuntu 16.04.

If you do not have a domain name, you can still secure your site for testing and learning with a [self-signed SSL certificate](#).

## Troubleshooting Nginx and Unicorn

If this last step does not show your application, you will need to troubleshoot your installation.

## Nginx Is Showing the Default Page Instead of the Django Application

If Nginx displays the default page instead of proxying to your application, it usually means that you need to adjust the `server_name` within the `/etc/nginx/sites-available/myproject` file to point to your server's IP address or domain name.

`server_name` within the `/etc/nginx/sites-available/myproject` file to point to your server's IP address or domain name.

Nginx uses the `server_name` to determine which server block to use to respond to requests. If you are seeing the default Nginx page, it is a sign that Nginx wasn't able to match the request to a server block explicitly, so it's falling back on the default block defined in `/etc/nginx/sites-available/default`.

The `server_name` in your project's server block must be more specific than the one in the default server block to be selected.

## Nginx Is Displaying a 502 Bad Gateway Error Instead of the Django Application

A 502 error indicates that Nginx is unable to successfully proxy the request. A wide range of configuration problems express themselves with a 502 error, so more information is required to troubleshoot properly.

The primary place to look for more information is in Nginx's error logs. Generally, this will tell you what conditions caused problems during the proxying event. Follow the Nginx error logs by typing:

```
$ sudo tail -F /var/log/nginx/error.log
```

Now, make another request in your browser to generate a fresh error (try refreshing the page). You should see a fresh error message written to the log. If you look at the message, it should help you narrow down the problem.

You might see some of the following message:

**connect() to unix:/home/sammy/myproject/myproject.sock failed (2: No such file or directory)**

This indicates that Nginx was unable to find the `myproject.sock` file at the given location. You should compare the `proxy_pass` location defined within `/etc/nginx/sites-available/myproject` file to the actual location of the `myproject.sock` file generated in your project directory.

If you cannot find a `myproject.sock` file within your project directory, it generally means that the `gunicorn` process was unable to create it. Go back to the [section on checking for the Gunicorn socket file](#) to step through the troubleshooting steps for Gunicorn.

**connect() to unix:/home/sammy/myproject/myproject.sock failed (13: Permission denied)**

This indicates that Nginx was unable to connect to the Gunicorn socket because of permissions problems. Usually, this happens when the procedure is followed using the root user instead of a `sudo` user. While the Gunicorn process is able to create the socket file, Nginx is unable to access it.

This can happen if there are limited permissions at any point between the root directory (`/`) the `myproject.sock` file. We can see the permissions and ownership values of the socket file and each of its parent directories by passing the absolute path to our socket file to the `namei` command:

```
$ namei -nom /home/sammy/myproject/myproject.sock
```

Output

```
f: /home/sammy/myproject/myproject.sock
```

```
drwxr-xr-x root root /
```

```
1
```

```
drwxr-xr-x root root none
drwxr-xr-x sammy sammy sammy
drwxrwxr-x sammy sammy myproject
srwxrwxrwx sammy www-data myproject.sock
```

The output displays the permissions of each of the directory components. By looking at the permissions (first column), owner (second column) and group owner (third column), we can figure out what type of access is allowed to the socket file.

In the above example, the socket file and each of the directories leading up to the socket file have world read and execute permissions (the permissions column for the directories end with `r-x` instead of `---`). The Nginx process should be able to access the socket successfully.

If any of the directories leading up to the socket do not have world read and execute permission, Nginx will not be able to access the socket without allowing world read and execute permissions or making sure group ownership is given to a group that Nginx is a part of. For sensitive locations like the `/root` directory, both of the above options are dangerous. It's better to move the project files outside of the directory, where you can safely control access without compromising security.

## Django Is Displaying: "could not connect to server: Connection refused"

One message that you may see from Django when attempting to access parts of the application in the web browser is:

```
OperationalError at /admin/login/
could not connect to server: Connection refused
Is the server running on host "localhost" (127.0.0.1) and accepting
TCP/IP connections on port 5432?
```

This indicates that Django is unable to connect to the Postgres database. Make sure that the Postgres instance is running by typing:

```
$ sudo systemctl status postgresql
```

If it is not, you can start it and enable it to start automatically at boot (if it is not already configured to do so) by typing:

```
$ sudo systemctl start postgresql
$ sudo systemctl enable postgresql
```

If you are still having issues, make sure the database settings defined in the `~/myproject/myproject/settings.py` file are correct.

## Further Troubleshooting

For additional troubleshooting, the logs can help narrow down root causes. Check each of them in turn and look for messages indicating problem areas.

The following logs may be helpful:

- Check the Nginx process logs by typing: `sudo journalctl -u nginx`
- Check the Nginx access logs by typing: `sudo less /var/log/nginx/access.log`
- Check the Nginx error logs by typing: `sudo less /var/log/nginx/error.log`

- Check the Gunicorn application logs by typing: `sudo journalctl -u gunicorn`

As you update your configuration or application, you will likely need to restart the processes to adjust to your changes.

If you update your Django application, you can restart the Gunicorn process to pick up the changes by typing:

```
$ sudo systemctl restart gunicorn
```

If you change `gunicorn` systemd service file, reload the daemon and restart the process by typing:

```
$ sudo systemctl daemon-reload
$ sudo systemctl restart gunicorn
```

If you change the Nginx server block configuration, test the configuration and then Nginx by typing:

```
$ sudo nginx -t && sudo systemctl restart nginx
```

These commands are helpful for picking up changes as you adjust your configuration.

## Conclusion

In this guide, we've set up a Django project in its own virtual environment. We've configured Gunicorn to translate client requests so that Django can handle them. Afterwards, we set up Nginx to act as a reverse proxy to handle client connections and serve the correct project depending on the client request.

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.

By: Justin Ellingwood

Upvote (57)

 Subscribe

## New on DigitalOcean: Cloud Firewalls

Secure your infrastructure and define what services are visible on your Droplets. Cloud Firewalls are free and perfect for staging and production deployments.

[MORE ON CLOUD FIREWALLS](#)

### Related Tutorials

[How To Create a Status Page with Cachet on Debian 8](#)

How To Add the log Module to Nginx on Debian 8







How to Use Full-Text Search in PostgreSQL on Ubuntu 16.04

How To Deploy a Laravel Application with Nginx on Ubuntu 16.04

How To Install and Configure Drone on Ubuntu 16.04

---

## 128 Comments

**B** *I*      

Leave a comment...

Log In to Comment

[Len](#) May 20, 2016

1 Found this super useful. Can you show the steps in this example to use uWSGI instead of Gunicorn?

---

jellingwood **MOD** May 20, 2016

- 2 @Len Hello. We don't have a guide that exactly meets that criteria, but [this one](#) is pretty close. You should be able to get that up and running and use Postgres for your database using the steps here.



#### How To Serve Django Applications with uWSGI and Nginx on Ubuntu 16.04

Django is a powerful web framework that can help you get your Python application or website off the ground. 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...

---

nymweb May 22, 2016

- 0 Me sirvio mucho para un proyecto que necesitaba. Sin duda el mejor tutorial que hay en internet.

---

JakeWatson June 5, 2016

- 3 Thanks for the tutorial Justin, I managed to get Nginx working after a few hiccups. If you are a idiot like me and make any mistakes on the the Gunicorn Service file and then enable Gunicorn, you'll likely get a Nginx default page/Bad Gateway 502 response. To get the Django default page and admin site back, you have to type:

```
sudo systemctl daemon-reload
sudo systemctl start gunicorn
sudo systemctl enable gunicorn
sudo systemctl restart nginx
```

*edited by jellingwood*

---

jeremt October 30, 2016

- 0 [deleted]



abprobe88bcd7d6 November 17, 2016

1 The first command should read:

```
sudo systemctl daemon-reload
```

JakeWatson June 18, 2017

o Thanks, my bad, can't edit original comment I'm afraid.

jellingwood MOD June 19, 2017

o @JakeWatson I fixed it up for you. Thanks for the input!

antkarvelas June 10, 2016

o Very useful, but it's not working for me. While the

```
gunicorn --bind 0.0.0.0:8000 myproject.wsgi:application
```

and running the Django server work just fine, using the setup of this tutorial results in either the "It worked!" default Django page or a page informing me of

```
RuntimeError at /
Model class jokes_app.models.Joke doesn't declare an explicit app_label and isn't in an application in INSTALLED_APPS.
Request Method: GET
Request URL: http://46.101.181.13/
Django Version: 1.9.7
Exception Type: RuntimeError
Exception Value:
Model class jokes_app.models.Joke doesn't declare an explicit app_label and isn't in an application in INSTALLED_APPS.
Exception Location: /home/antoni4040/django_project/django_project_env/lib/python3.5/site-packages/django/db/models/base.py in __new__, line 102
Python Executable: /home/antoni4040/django_project/django_project_env/bin/python3
Python Version: 3.5.1
Python Path:
['/home/antoni4040/django_project/django_project',
'/home/antoni4040/django_project/django_project_env/bin',
'/home/antoni4040/django_project/django_project_env/lib/python3.5.zip',
'/home/antoni4040/django_project/django_project_env/lib/python3.5',
'/home/antoni4040/django_project/django_project_env/lib/python3.5/plat-x86_64-linux-gnu',
'/home/antoni4040/django_project/django_project_env/lib/python3.5/lib-dynload',
'/usr/lib/python3.5',
'/usr/lib/python3.5/plat-x86_64-linux-gnu',
'/home/antoni4040/django_project/django_project_env/lib/python3.5/site-packages']
Server time: Fri, 10 Jun 2016 06:04:10 +0000
```

Any ideas?

---

[seanmavley](#) July 1, 2016

1 this is your actual error, not the fault of gunicorn:

"Model class `jokesapp.models.Joke` doesn't declare an explicit `app_label` and isn't in an application in `INSTALLED_APPS`."

So you can start debugging the problem from there.

---

[nerdlifejohn](#) June 14, 2016

1 Great write up I got it working in one shot, I have 2 questions though.

Q1: Does 'systemctl' replace the need for supervisor?

Q2: Does this setup allow for multiple domains/gunicorn.socks to be hosting on the same machine?

---

[ahmedalshamary](#) June 19, 2016

0 I followed the instructions and created everything. However when I try to open it up in the end I get a 502 Bad Gateway error. I also noticed that I don't have a `myproject.sock` file created. Where would you recommend I look to try and fix the problem. Thanks

---

[jarjar](#) July 12, 2016

0 Refer to [JakeWatson's comment](#)



#### **How To Set Up Django with Postgres, Nginx, and Gunicorn on Ubuntu 16.04**

Django is a powerful web framework that can help you get your Python application or website off the ground. 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...

---

[kalakak](#) August 2, 2016

1 i have tried all, but could not solve the 502 bad gateway problem

- 1 I'm having the same issue as [@ahmedralshamary](#). I tried the procedure mentioned in [JakeWatson's comment](#) but the first command gives me an error.

`sudo systemctl daemon reload` gives the error:

Unknown operation daemon.



#### How To Set Up Django with Postgres, Nginx, and Gunicorn on Ubuntu 16.04

Django is a powerful web framework that can help you get your Python application or website off the ground. 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...

[abprobe88bcd7d6](#) November 17, 2016

- 1 OK. The problem with the first command in [Jake Watson's comment](#) is that it's missing a hyphen, and should read:  
`sudo systemctl daemon-reload`

However, even executing these four commands, I still do not have a `.sock` file in my project directory. Please help



#### How To Set Up Django with Postgres, Nginx, and Gunicorn on Ubuntu 16.04

Django is a powerful web framework that can help you get your Python application or website off the ground. 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...

[marcduplessis1](#) February 13, 2017

- 1 [@ahmedralshamary](#) [@abprobe88bcd7d6](#) [@kalakak](#) [@bobanda](#)

I was also getting a 502 error all the time. I found a few ways on how to check where the error is:

1. You can run `sudo systemctl status gunicorn`. This gives a status on gunicorn, from this I found that my settings.py was using environmental variable set in my virtual environments activate file and gunicorn doesn't know them.
2. running `gunicorn --log-file=- subby.wsgi:application` This identified a spelling mistake I had in settings.py it will also give other useful information.

After fixing the errors I ran `sudo systemctl status gunicorn` again and saw it was successful and a project.sock was created

[orduani](#) May 23, 2017

- 0 [@marcduplessis1](#) answer helps A LOT!

[kalakak](#) August 2, 2016

- 1 hey did you able to solve it, if you did please help

---

lgs June 26, 2016

1 There is an inconsistency in the directions for setting up gunicorn.service.

It goes from

```
After=network.target
```

to

```
After=networking.target
```

then back to

```
After=network.target
```

---

jellingwood MOD June 27, 2016

1 @lgs: Good catch! Should be fixed up now. Thanks for pointing that out.

---

seanmavley July 2, 2016

1 Tutorial is great.

Maybe what you should be aware of is this:

To be able to serve static files from your project *properly*, doing below will save you some hours of head scratching.

```
root /path/to/project/project_name;
```

```
location /static/ {  
    alias /path/to/project/project_name/static;  
}
```

Assuming /static is where you store your statics, and you have this in your django:

```
STATICROOT = os.path.join(BASEDIR, 'static')
```

---

sidp July 2, 2016

1 What do the myproject directories look like? I'm trying to implement this with a project I have already created but it's difficult to know exactly what directories to use when I don't know what the tree looks like.

1 @sidp: The project structure used for the guide is just the basic project structure given to you by the `django-admin startproject` command.

If you type the commands outlined in the tutorial:

```
$ mkdir ~/myproject
$ cd ~/myproject
$ django-admin.py startproject myproject .
```

You would get a standard directory tree that looks like this:

```
myproject/
├── manage.py
└── myproject
    ├── __init__.py
    ├── settings.py
    ├── urls.py
    └── wsgi.py
```

If you are having trouble matching your project's structure to the tutorial, it might be easiest to go through the guide using the dummy app first, to get familiar with all of the pieces, and then go back to set up your real project.

o Hi,

I'm trying to set up the server on a local netowrok, not public IP, I followed all steps but without suces, I bealive it has to be more on the service configuration..

Any suggestions how the configuration would be on a local network?

---

bobanda July 11, 2016

o Hi, thanks for a great tutorial!

However I can't make it to work for some reason.

I can see that `nginx` is running (when I enter url in the browser).

Also, I checked the `gunicorn.status` and I get this response:

```
gunicorn.service - gunicorn daemon
Loaded: loaded (/etc/systemd/system/gunicorn.service; enabled; vendor preset: enabled)
Active: active (running) since Mon 2016-07-11 17:51:33 EDT; 6min ago
Main PID: 4344 (gunicorn)
CGroup: /system.slice/gunicorn.service
├─4344 /home/django/venvs/myenv/bin/python3 /home/django/venvs/myenv/bin/gunicorn --bind unix:/home/django/myapp/myapp.sock
└─4351 /home/django/venvs/myenv/bin/python3 /home/django/venvs/myenv/bin/gunicorn --bind unix:/home/django/myapp/myapp.sock

Jul 11 17:51:33 16.04.gunicorn.django systemd[1]: Stopped gunicorn daemon.
Jul 11 17:51:33 16.04.gunicorn.django systemd[1]: Started gunicorn daemon.
Jul 11 17:51:34 16.04.gunicorn.django gunicorn[4344]: [2016-07-11 17:51:34 -0400] [4344] [INFO] Starting gunicorn 19.6.0
Jul 11 17:51:34 16.04.gunicorn.django gunicorn[4344]: [2016-07-11 17:51:34 -0400] [4344] [INFO] Listening at:
unix:/home/django/myapp/myapp.sock
Jul 11 17:51:34 16.04.gunicorn.django gunicorn[4344]: [2016-07-11 17:51:34 -0400] [4344] [INFO] Using worker: sync
Jul 11 17:51:34 16.04.gunicorn.django gunicorn[4344]: [2016-07-11 17:51:34 -0400] [4351] [INFO] Booting worker with pid: 4351
Jul 11 17:53:02 16.04.gunicorn.django systemd[1]: Started gunicorn daemon.
```

So, everything seems like it is running. But, when I enter URL on port 8000, it doesn't render anything.

Where can I actually set the port where it will run it?

When I run it like this:

```
gunicorn --bind 0.0.0.0:8000 myapp.wsgi:application
```

then it works (on port 8000).

Can you help me set the port in the config somehow? And can you explain on which port is running if you don't set any port? Thanks!

---

bobanda July 11, 2016

o I figured out the problem.

The problem was because I didn't change `server_name` in `nginx` config

```
server_name server_domain_or_IP;
```

---

[jarjar](#) July 11, 2016

- o Hi, thank you for making this step-by-step guide. You rock! I have one question however. I have my unicorn.service configured as shown in the guide and 'myproject' is in sites-enabled and sites-enabled and I restarted nginx. Now whenever I access my IP, it brings me to the 'hello-world' django page -- the one with the "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!" message displayed. I uploaded my apps (myapp1, myapp2) files to '/home/sammy/myproject', I configured 'myproject' urls.py and when accessing my ip the only url patterns available are '/admin' and the django landing page mentioned before. I executed the './manage.py makemigrations' and './manage.py migrate' and './manage.py collectstatic'

```
myproject/
├── manage.py
├── myprojectenv
├── myproject
│   ├── __init__.py
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
├── myapp1
└── myapp2
```

---

[matthew7821f27b](#) July 14, 2016

- 1 Hello, I have followed your instructions word for word. Although I am getting strange 502 bad gateway errors. I checked the error logs of nginx and it says that under path/to/my/project/dir/project.sock (no file or directory exists). I have looked all over online and was wondering if anyone has a solution.

---

[kenschnall](#) July 28, 2016

- o Hello, I have the same issue.

I am getting a **502 Bad Gateway** error.

When checking the nginx error logs with `sudo tail -f /var/log/nginx/error.log` I see this:

*connect() to unix:/home/USERNAME/PROJECTNAME/PROJECTNAME.sock failed (2: No such file or directory) while connecting to upstream*

This tutorial did not tell me to create a .sock file inside my Django project. How can we fix this error?

**Edit:**

This does not our issue but helps me understand what the .sock files represents

[Stack Overflow](#)



---

yoperodaniel August 20, 2016

- o I was able to find a work around by editing /etc/systemd/system/gunicorn.service

```
ExecStart=/home/sammy/myproject/myprojectenv/bin/gunicorn --workers 3 --bind
unix:/home/sammy/myproject/myproject.sock myproject.wsgi:application
```

with

```
ExecStart=/home/sammy/myproject/myprojectenv/bin/gunicorn --workers 3 --bind 0.0.0.0:8000 myproject.wsgi:application
```

I think the unix socket is not working because of some sort of permissions issues that do not allow nginx to connect to the unix:/home/sammy/myproject/myproject.sock

I will post back if I get to the bottom of it.

---

rdkaiser August 20, 2016

- 1 1. Make sure the socket is readable/writable by the nginx worker process (probably www-data).

```
$ chgrp www-data myproject.sock && chmod g+rw myproject.sock
```

2. Make sure the directory containing the socket (and all parent directories) is/are executable by that process.

```
$ chmod +x /home/sammy /home/sammy/myproject
```

3. You also might want to set the SETGID bit on the socket directory.

```
$ chmod g+s /home/sammy/myproject
```

Personally, I would not keep the socket and other runtime files in the project directory. /var/local/run/myproject is a good place.

0

Make sure the socket is readable/writable by the nginx worker process (probably www-data).

What if the sock file doesn't exist? What could be the reasons why wsgi doesn't create this file? I've followed all the instructions.

This command works:

```
gunicorn --bind 0.0.0.0:8000 myapp.wsgi:application
```

And I had to change "/etc/nginx/sites-available/myproject" from

```
listen 80;
```

to

```
listen 8000;
```

That's all the difference with this tutorial, and I still have 502 error.

Am I the only one?

---

taushifkhan November 4, 2016

- o I am facing the same problem. Could you able to find a work around yet ?

---

BenDevelopment November 4, 2016

- o @taushifkhan I don't remember exactly what I have done to correct this problem but I can share my notes maybe it can help:

Create a Gunicorn systemd Service File:

gunicorn file: /etc/systemd/system/projectnameproduction.service

```
[Unit]
Description=gunicorn project_name production daemon
After=network.target

[Service]
User=ben
Group=www-data
WorkingDirectory=/projects/project_name/backend/production/project_name_rest
ExecStart=/projects/project_name/backend/production/project_name_rest/env/bin/gunicorn --workers 3 --bind
unix:/projects/project_name/backend/production/run/project_name_rest.sock project_name_rest.wsgi:application

[Install]
WantedBy=multi-user.target
```

nginx file: /etc/nginx/sites-available/projectnameproduction

```
server {
    listen 8825;
```

```
server_name 139.59.150.116;

location = /favicon.ico { access_log off; log_not_found off; }
location /static/ {
    root /projects/project_name/backend/production/project_name_rest;
}

location / {
    include proxy_params;
    proxy_pass http://unix:/projects/project_name/backend/production/run/project_name_rest.sock;
}
}
```

```
sudo ln -s /etc/nginx/sites-available/project_name_production /etc/nginx/sites-enabled
mkdir /projects/project_name/backend/production/run
sudo chown ben:www-data /projects/project_name/backend/production/run/
sudo systemctl start project_name_production
sudo systemctl enable project_name_production
sudo systemctl restart nginx
```

errors:

sock file not found:

change run folder access (this is the folder where sock file will be created by wsgi):

```
sudo chown ben:www-data /projects/projectname/backend/production/run/
```

kill gunicorn daemons:

```
pskill gunicorn
```

reload daemons:

```
sudo systemctl daemon-reload
```

start gunicorn:

```
""sudo systemctl start projectname_production""
```

PS: This reply form is the worst I've ever seen.

---

[kenschnall](#) November 4, 2016

- o [@taushifkhan](#) I gave up starting from scratch and used my existing Python 3 Django code like sentdex did in his youtube video:

[https://www.youtube.com/watch?v=Y-CT\\_l1dnVU](https://www.youtube.com/watch?v=Y-CT_l1dnVU)

Seriously though just use the one-click droplet for Django, go through the initial server setup

(<https://www.digitalocean.com/community/tutorials/initial-server-setup-with-ubuntu-14-04>) and then follow his youtube video

- 1 • First, go back to using a socket; also, ensure gunicorn starts before nginx:

```
/etc/systemd/system/gunicorn.service
```

```
[Unit]
...
Before=nginx.service
...

[Service]
...
ExecStart=/home/sammy/myproject/myprojectenv/bin/gunicorn --bind unix:/full/path/to/socket_directory/myproject.sock
myproject.wsgi:application
...
```

```
/etc/nginx/sites-available/myproject
```

```
upstream myproject {
    server unix:/full/path/to/socket_directory/myproject.sock;
}

server {
    listen 80;
    ...
    location / {
        ...
        proxy_pass http://myproject;
        ...
    }
    ...
}
```

- If that doesn't work, use the following snippets to set the proper permissions.

```
$ iterdir () {
    declare -n _dir=dir &&
    _dir="${1:-`dirname "${_dir:--}"`}" &&
    ! egrep -q '^(/)$' <<<"$_dir" || return 1
}
```

```
$ read -ep "sockdir (/full/path/to/socket_directory) : " sockdir
```

```
# stat "${sockdir:?}" >/dev/null && chown ${myuser:=$(stat -c %U "$sockdir")}:www-data "$sockdir" && iterdir "$sockdir" &&
```

- Restart gunicorn, then restart nginx:

```
# systemctl restart gunicorn && systemctl restart nginx
```

---

[kalakak](#) August 2, 2016

o did you able to solve it? need help

---

[bhartendu2007](#) July 30, 2016

o Hi,

Thanks for the great tutorial, but i have found myself stuck at the following step and can't fmove forward with the setup.

Finally, you can test our 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

When I visit my server's domain name it only says **This site can't be reached.**

I am using aws ec2 server with elastic IP.

While the terminal shows that there are no issue and the server is running on 0.0.0.0:8000, I can't see the django default page on my domain.

Please do let me know if any more information required from my end.

Thanks

---

[jakubowskiigor](#) August 7, 2016

o Superb Tutorial! Great for new digitalocean users. Any chance someone knows a clue how to make a domain name display the content of the webpage after one has taken all the steps described above? After finishing the tutorial I can successfully see the django project @ the ip of the server but not my domain. Do I have to change the nginx settings somehow?

---

[joeklinck](#) August 12, 2016

o If you decide to not use a virtualenv then you will need to alter "ExecStart" in "/etc/systemd/system/gunicorn.service". Just run "which gunicorn"(that will tell you the file path where gunicorn is stored) and use that as the first part of the "ExecStart variable". For example, on a project of mine I used this:

```
ExecStart=/home/joe/.local/usr/bin/gunicorn --workers 3 --bind unix:/home/joe/mysite/mysite.sock mysite.wsgi:application
```

And remember when making any changes to the "gunicorn.service" file you need to reload gunicorn with these commands:

```
sudo systemctl daemon-reload
sudo systemctl start gunicorn
sudo systemctl enable gunicorn
```

And then you need to restart nginx:

```
sudo systemctl restart nginx
```

---

[miguelghz](#) September 1, 2016

o Great article, it helped me a lot in my project!  
Best regards! [Miguel A. Gómez](#)

---

ArdaMavi September 13, 2016

0 How can I add second django app in gunicorn.service ?

---

ArdaMavi September 17, 2016

1 Pls help me

502 Bad Gateway  
nginx/1.10.0 (Ubuntu)

---

adrianvalenz October 19, 2016

0 Yea I need help too!

---

adrianvalenz October 20, 2016

0 did you figure it out?

---

AleXandrO November 23, 2016

1 Show as your gunicorn service, maybe it could help  
sudo systemctl status gunicorn.service

---

sneakycr0w November 27, 2016

1 I'm having the same issue as these two. Here's what I get when I run that command:

- gunicorn.service - gunicorn daemon

Loaded: loaded (/etc/systemd/system/gunicorn.service; enabled; vendor preset: enabled)

Active: failed (Result: exit-code) since Sun 2016-11-27 03:16:02 UTC; 4min 20s ago

Process: 1244 ExecStart=/home/zach/myproject/myprojectenv/bin/gunicorn --workers 3 --bind

unix:/home/zach/myproject/myproject.sock myproject.wsgi:application (code=exited, status=217/USER)

Main PID: 1244 (code=exited, status=217/USER)

Nov 27 03:16:02 django-gunicorn-nginx-letsencrypt systemd[1]: Started gunicorn daemon.

Nov 27 03:16:02 django-gunicorn-nginx-letsencrypt systemd[1]: gunicorn.service: Main process exited, code=exited, status=217/USER

Nov 27 03:16:02 django-gunicorn-nginx-letsencrypt systemd[1]: gunicorn.service: Unit entered failed state.

Nov 27 03:16:02 django-gunicorn-nginx-letsencrypt systemd[1]: gunicorn.service: Failed with result 'exit-code'.

Load More Comments



This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.



Copyright © 2017 DigitalOcean™ Inc.

[Community](#) [Tutorials](#) [Questions](#) [Projects](#) [Tags](#) [Newsletter](#) [RSS](#) 

[Distros & One-Click Apps](#) [Terms, Privacy, & Copyright](#) [Security](#) [Report a Bug](#) [Get Paid to Write](#) [Shop](#)

Sign up for our newsletter. Get the latest tutorials on SysAdmin and open source topics. 

Sign Up