

Django Walkthrough and AWS Setup compiled by Andy Boyle.

Source URL: <http://www.andymboyle.com/2011/01/29/step-one-setting-up-your-ubuntu-server-on-amazon-web-services/>

# Step One: Setting up your Ubuntu server on Amazon Web Services

*For Step Two, go [here](#).*

In this post I will walk you through how to set up a web server on Amazon Web Services with Ubuntu. I will also explain some incredibly bare bones basic stuff about the interwebs, as I feel this knowledge is lacking in newsrooms today. It's the first part in a series of walkthroughs on how I set up a recent project in Django using Ubuntu, Postgresql, nginx, uWSGI and Varnish.

This is geared more toward the journalist who wants to learn some basic development chops, so pardon me if I try to make this not super technical. And in no way is that because I only half know what I'm talking about. In no way.

So before I walk you through setting up a server, let me sort of explain how the Internet works.

## HOW THE INTERNET WORKS

Say you want to visit a website. Perhaps it's [www.andymboyle.info](http://www.andymboyle.info). You type the address into your browser, and somehow, magically, a screen appears with text, an image of a beautiful man and awesome music plays in the background. This is quite similar to the process of you looking up a new bar to visit and then asking a bartender for a beer.

When looking up the location of a bar, sometimes people search a phone book (if this was the 1970s). You find the physical location of the bar, 1000 1st St., and then you showed up. You would walk in, sit down at the counter, and ask the bartender for a **Yuengling**, because you are a True Patriot.

On the Internet, when you visit a website, your browser is searching a digital phonebook to find the physical location of the website, which resides on a server somewhere. It finds the address of the server — or IP address (Internet Protocol address) — and heads there. It asks the server to do something — in most cases, please fetch the HTML of the page (please give me a beer, bartender) — and the server does it. Then your page is displayed.

You now know how the Internet works. *You're welcome.*

## SETTING UP AN AWS INSTANCE

Now go to [aws.amazon.com](https://aws.amazon.com) and set up an account. For a quick walkthrough, [you can read this](#). After you've done that, sign in. Click on the EC2 tab. Look under "Service Health." It should list something like "Amazon EC2 (US East – N. Virginia)." You want to remember what it says in the first part of the parenthesis, as we're going to need that later. Mine is US East, but yours could be different.

Click on "Launch Instance," then the Community AMIs tab. We're going to be installing Ubuntu 10.04 LTS, which is the operating system for the physical server. Similar to how your computer needs an operating system, so does your physical server.

You need to search for the right AMI to use. An AMI is a copy of a server that someone already setup. Ubuntu keeps a bunch up and going and shows you where to [find them at this website](#). We will be using the 32-bit EBS version. So, for me I find US East 32-bit EBS's number is ami-a2fe04cb (as of me writing this). Search for that under the Community AMIs tab. Once it finds it, click "Select."

You want 1 instance, availability zone is no preference (feel free to change this stuff if you're more knowledgeable than me), instance type is small, click continue. Then click continue again. On the next screen, click on the spot to the right of "Name" and give this server a name. Perhaps "Django Awesome Project."

Create a key pair if you haven't yet and then download it. It should be something like "blah.pem." This is what you will use to login to your server later. On the next page, click default for security groups (unless you've already set stuff up yourself, feel free to use that). Then click launch. Give it a few minutes, but it should appear on your list of instances (on the left rail, click on "Instances").

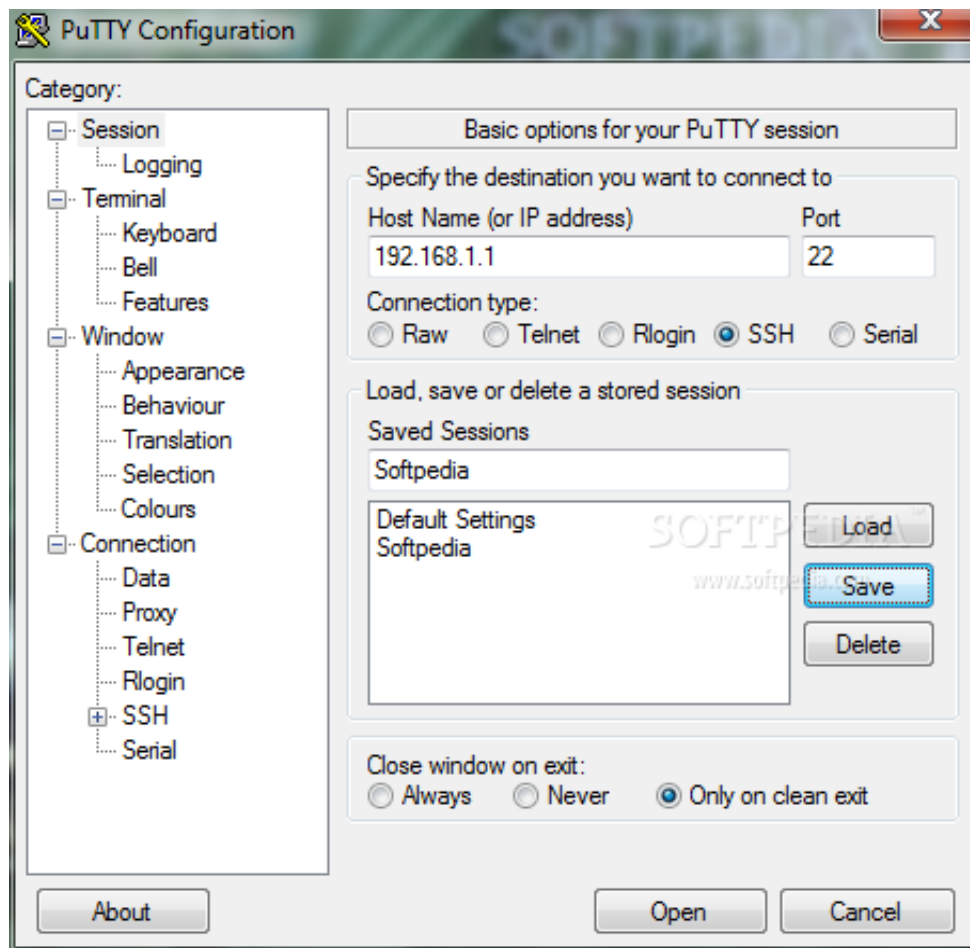
Click on "Django Awesome Project" or whatever you named it. A window should appear below the list of instances, and scroll down. You will see something called "Public DNS" and then some stuff listed after it. That's where your server lives — that's the address of the bar. If it said, for instance, ec2-40.10.10.10.compute-1.amazonaws.com, that's where your stuff lives. The 40.10.10.10 would be your IP address.

Now you need to log into your server. First I will explain how to do it with Windows and then with Apple OS X. If you're on a Linux box, you can probably follow along the OS X one. Also, you probably don't need to read this because you're obviously a superior human being.

## LOGGING ONTO YOUR SERVER WITH A

# WINDOWS TERMINAL

If you're on Windows, I'm sorry but that sucks. You need to **download PuTTY**, which is a terminal client for Windows. It works okay. Once you install that, open it.



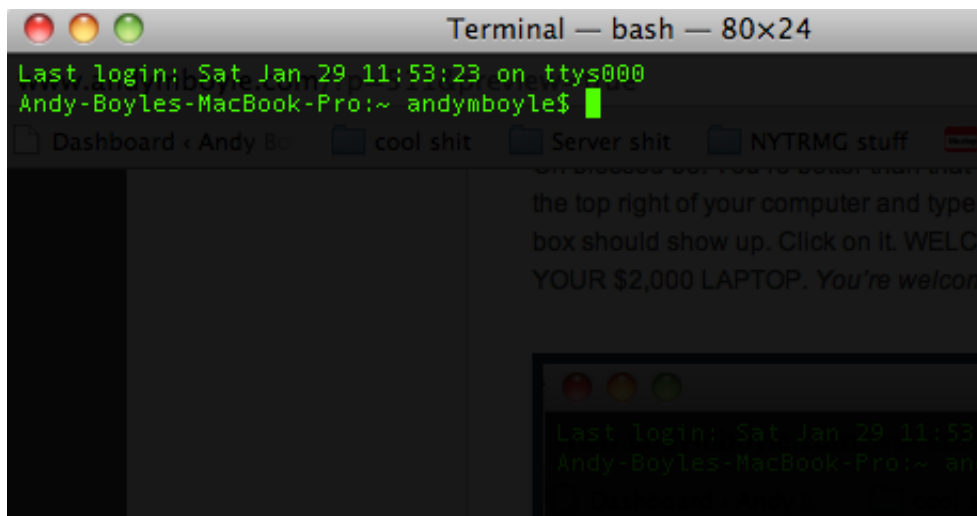
Under host name, put in what you found after “Public DNS,” which would be similar to ec2-40.10.10.10.compute-1.amazonaws.com and make sure under protocol “SSH” is marked. But in front of that put ubuntu@, which is the username, so altogether it looks like ubuntu@ec2-40.10.10.10.compute-1.amazonaws.com

On the left you should see a part that says SSH (you may need to click on Connection first). Then click on Kex. Find wherever it is you downloaded that key pair to and put that in there (it could have a .pem extension). **Read this and convert your .pem file to a .ppk**. Now, click Session again. Give this thing a name, perhaps “Awesome Django Project” under “Saved Session” and then click Save.

Now click load. A screen should pop up. It may ask you a question about “yes or no.” Say yes. And bam, you’re on the physical server. This is called a terminal window, and you are now awesome.

## LOGGING ONTO YOUR SERVER WITH MAC OS X

Oh blessed be. You’re better than that Windows guy. Click the magnifying glass on the top right of your computer and type in “Terminal.” An app that looks like a black box should show up. Click on it. WELCOME TO THE MOST POWERFUL PART OF YOUR \$2,000 LAPTOP. *You’re welcome, again.*



So. Remember that key pair you downloaded? You need to go to the directory it’s in. If it’s on your desktop, then you would need to type

```
cd desktop
```

If it’s in another folder, then cd to that folder name. I’m not going to teach you basic terminal commands. So. Deal with that on your own, please. Also: I believe AWS forces you to download a filename.pem. If you were given a .ppk file by a coworker like some of us instead of the .pem version, you’ll need to convert it into a .ssh file. You can read how to do that [here](#) if you have access to a Windows machine. I installed a program to do it on my Mac, but I can’t recall what it was called. It also borked some stuff up, so that sucked. And a word of advice: DON’T INSTALL MACPORTS.

Now, once you're in the directory with the file — now a .ssh file — you'll type this command:

```
ssh ubuntu@ec2-40.10.10.10.compute-1.amazonaws.com -i key_file_name.extension
```

You should get a screen that looks like this:

```
Welcome to Ubuntu!  
* Documentation:  https://help.ubuntu.com/  
  
System information as of Sat Jan 29 22:07:15 UTC 2011
```

And huzzah! Your server is up and running. In the next installment, we'll walk through setting up Django, uWSGI and nginx and how to configure it all.

Now on to **Step Two: Installing and setting up a basic Django project**

## Step Two: Installing and setting up a basic Django project

In this post I will walk you through how to set up a basic Django project on that **AWS instance we've already set up**.

As you remember, after logging in with your terminal application, you'll be at a prompt. What follows are some basic commands for setting up and installing various tools.

## SETTING UP YOUR DIRECTORIES

You may need to type "sudo" in front of most of these commands (without the quotes). That stands for "super user do," which allows you to execute commands without having to login as the root user. First, let's update the ubuntu we've installed. Enter each command individually:

```
apt-get update  
shutdown -r now
```

That last part will shut down your server and kick you off, so you'll need to log back in, like I showed you near the end of step one.

Next we will set up various directories that we will use later on. I prefer putting my django-projects in an /opt/ directory, as **Jeremy Bowers** always made me do. He is smarter than me and eats fancy pork products, so you should listen to him, too. Also, he explains his reasoning in [this blog post](#) (which I'll be referencing a lot in Step Six of this tutorial).

Again, remember you may need to type "sudo" in front of these commands, except the ones that are cd. Enter each command individually:

```
cd /opt/  
mkdir /opt/run/  
mkdir /opt/log/  
mkdir /opt/log/nginx/  
mkdir /opt/lock/  
mkdir /opt/django-projects/  
chmod -R 777 /opt/
```

The mkdir command makes a directory. The chmod -R 777 makes a directory read and writeable by all users, which is what you'll want when you're setting up your apps. What we've done is make the directory where your Django apps will live.

We've also added stuff for nginx, which will be our application server. This is our software server, which lives on a physical server running Ubuntu. Think of nginx as the bartender inside of a bar, using my analogy from Step One.

## INSTALLING A BUNCH OF SHIT

Now you'll want to enter this command as one line — yes, one line, so copy, past and hit enter — and it'll install a ton of stuff you'll need, and some stuff you may never use. But hey, you'll have it around and it doesn't really hurt anything.

```
apt-get install binutils python-psycopg2 python-setuptools libgeos-3.1.0 libgeos-  
c1 libgdal1-1.6.0 postgresql-8.4-postgis postgresql-server-dev-8.4 haproxy nginx  
memcached python2.6 python2.6-dev psmisc subversion git-core mercurial python-  
imaging locate ntp python-dateutil libxml2-dev libpcre3 libpcre3-dev
```

That will take a few minutes, and if it asks you any yes or no questions, always answer yes. Let's make sure python was installed. At your prompt type:

```
python
```

You should have some text pop up about the python version installed, followed by a >>> prompt. That means python is installed! Huzzah. You are now wonderful and can finally do amazing things with your life. Type this to exit:

```
exit()
```

You already have **PostgreSQL** installed from that long thing you copy and pasted up there. That will be your database backend. You could use MySQL, sure, but I don't. And besides, don't you want to be cool like me? Next let's install **Django**.

## INSTALLING DJANGO

Type this, remembering you may need to use sudo before each command:

```
easy_install django pysolr django-storages django-piston pywapi
```

I probably just blew your mind. I apologize. We just installed Django along with some other awesome tools. The easy\_install function was installed as part of the python setup tools you did when you copied that previous big long command up there.

Now, to make sure Django is installed, go to the python prompt — we did this earlier, remember? — by typing:

```
python
```

Then type at the >>> prompt each line separately and hit enter:

```
import django

print django.get_version()
```

It should spit out 1.2.4 (or whatever the most recent stable version of Django is). If so, awesome. If not, back up a few steps and retry to easy\_install Django. Then exit, like I showed you a few graphs up, with exit()

Next, go to the directory we want to install our Django app by typing this this:

```
cd /opt/django-projects/
```

We shall now make a basic Django application. I will walk you through coding a simple app in Step Three, but for now we will just set up our project and app.

## SETTING UP A BASIC DJANGO PROJECT/APP

The project we will make shall keep track of fires in your local community that are reported on, an app that Adrian Holovaty, one of the inventors of Django, **mentioned newspapers should be doing more than four years ago**.

So first we need to make a project directory. We shall name it firetracker. You create the project this way:

```
django-admin.py startproject firetracker
```

Now type:

```
ls
```

You should see a new directory called firetracker, and then cd into it (cd firetracker).



Type ls again and you'll see a bunch of files. I will explain what they are in a later post. But this is the project directory. Now you will make an application used this command:

```
django-admin.py startapp fires
```

Type ls again. See the fires directory? cd into it. You should see files like models.py, views.py, \_\_init\_\_.py and maybe tests.py. Congratulations! YOU'VE SET UP THE BASICS OF YOUR FIRST DJANGO APP!

Pat yourself on the back. You've just done some commands that might scare a regular mortal. Updating a server, downloading and installing new software, this fancy python command. You are on your way to becoming a regular Django Knight, and I salute you.

I highly suggest you go and drink a beer right now before **moving on to Step Three: Connecting to your server with an FTP client.**

## 4 Responses to “Step Two: Installing and setting up a basic Django project”

---



**gotoplanb** on January 30th, 2011

For setting up directories, you can just go with:

```
mkdir /opt/{run,log,log/nginx,lock,django-projects}  
chmod -R 777 /opt/
```



**gotoplanb** on January 30th, 2011

And if you're gonna install all the shit together, go ahead and put a “-y” switch at the end so it just starts instead of prompting you to say yes to the installs.



**gotoplanb** on January 30th, 2011

To install Django, I instead started off with:

```
$easy_install pip
```

Then I used the pip package manager for the same libs you installed. All seemed to go fine for me.

---

## Step Three: Connecting to your server with an FTP client

Now that you've **set up your Ubuntu server, installed Django and set up your app**, now it's time to actually code the damn thing. But first you need to learn how to FTP into your server with an FTP client.

What's that, you ask? It's the thing you use to connect to the server, edit files, upload stuff, etc. It's sweet.

Oh, also: I've also decided at this point that I'm not going to focus at all on setting stuff up on a Windows computer. Sorry. It's just too damn difficult. Either install Linux, buy a Mac or pick something else to do.

## LOGGING IN WITH A FTP CLIENT

I'm being lazy, so I'm going to just show you how to connect with FTP. Yes, everyone will tell you that SFTP is better. It is safer. It is more secure. But dammit, let me be lazy right now. You can learn more about **using SFTP with AWS and Coda here**.

**We're going to use vsftpd.** First, ssh into your server. Then type this to download and install vsftpd:

```
sudo apt-get install vsftpd
```

Then you're going to need to edit your permissions and add a user.

So let's create a user.

```
sudo adduser somename
```

It will then ask you to enter in a password — which you need to remember — and then other information.

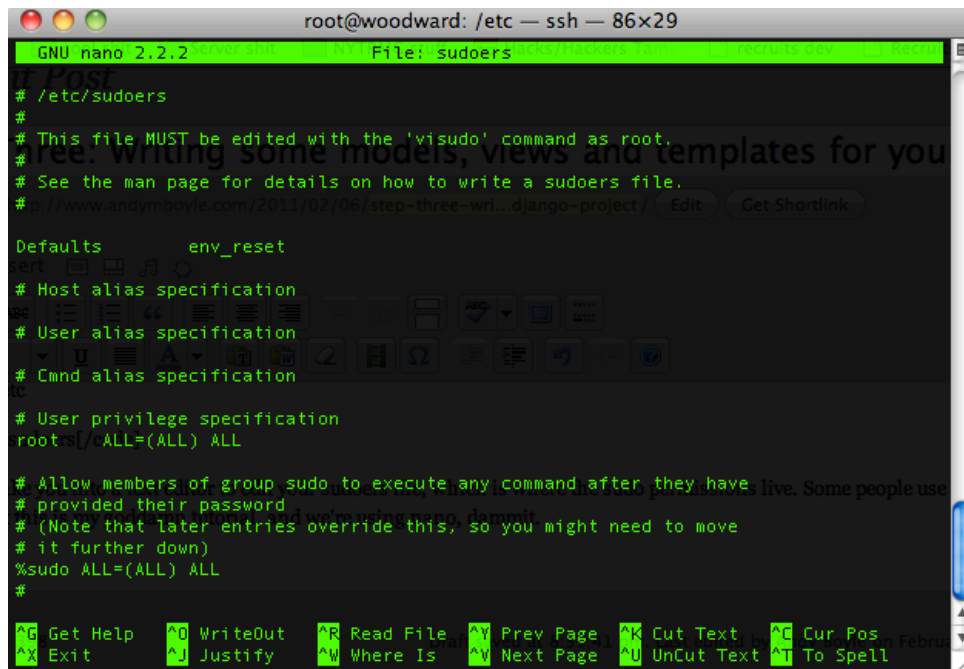
The password is the only thing you should really care about, feel free to hit enter on everything else.

Next we're going to want to give that person sudo power, so you can stop logging in as ubuntu. Do this:

```
cd /etc  
  
sudo nano sudoers
```

This will take you into a text editor to edit your sudoers file, which is where the sudo permissions live. Some people use different text editors, but this is my goddamn tutorial, and we're using nano, dammit.

Here's what your screen looks like:



Now, with your arrow keys, scroll down to underneath root. You're going to add that username you added previously, and then type `ALL=(ALL) ALL` after it, so basically it's copying what the root has.

```
somename    ALL=(ALL) ALL
```

Type Ctrl + O to save, then Ctrl + X to exit. Bam. You just made a user. You're super awesome.

Now let's edit your permissions. [Go here for more information on this.](#)

```
sudo nano vsftpd.conf
```

Now, for security reasons (I say this as I teach you how to use FTP, ha ha ha), change this line

```
anonymous_enable=YES
```

to this

```
anonymous_enable=NO
```

Then uncomment this line by deleting the # from them (that is how you uncomment stuff, by the way):

```
write_enable=YES
```

Then Ctrl + O followed by Ctrl + X. Next, enter in this command:

```
sudo /etc/init.d/vsftpd start
```

Now you can FTP in using **FileZilla**, which is what we're going to use. I would suggest you **go and buy Coda**, but you don't have to. I do. As do other cool people.

So open up FileZilla after you download and install it. Click the top left icon above the word "Host." Click new site, give it a name. For the Host field, you need to put your IP address of your AWS instance. You find that by looking at AWS server name, which was something like ec2-**40.10.10.10**.compute-1.amazonaws.com. In this instance, the IP would be 40.10.10.10.

User is the username you created up there, and password is the password. Server type should be FTP. Click on Transfer Settings, and change it to “Active.” Now click Connect.

This should look familiar. It's the structure of the files. Voila. You have now set up vsftpd, learned the very basics of using nano to edit stuff in the terminal and FTPd to your server. Next you will learn how to set up your models.py file. **So move on to Step Four.**

## Step Four: Writing your Django models

Now that you can FTP into your server and have a database set up, it's time for the fun to begin: Building your first project!

As I mentioned in **Step Two**, we're building FireTracker, your fancy Tracker Of Fires app. And with that, you need to know how Django works in order to build the damn thing.

Django builds what are known as models-view-controller, or MVC, applications. They're all the rage, and allow people to work on different parts at the same time for more efficiency. **Ruby on Rails** is another MVC that people use. I know Django better, and that's why we're building this app in Django.

Your **models.py** file is where you set up your data architecture, or in simpler terms, it's sort of where you set up spreadsheets with field names and what type of data will go in it (numbers, words, etc.).

Your **views.py** is where you tell it “hey, look at the models so we can tell the template how to spit stuff out.” And then the **template** is where your HTML and design lives, and it talks to the view and spits stuff out.

So let's talk about what we want this app to do.

## WHAT WILL THIS DAMN THING DO

You want to track fires that occur in your coverage area, right? Well, as **Adrian Holovaty once talked about:**

*For example, say a newspaper has written a story about a local fire. Being able to read that story on a cell phone is fine and dandy. Hooray, technology! But what I really want to be able to do is explore the raw facts of that story, one by one, with layers of attribution, and an infrastructure for comparing the details of the fire — date, time, place, victims, fire*

*station number, distance from fire department, names and years experience of firemen on the scene, time it took for firemen to arrive — with the details of previous fires. And subsequent fires, whenever they happen.*

So think of it this way. You get a report of a fire from the scanner. You run out to cover it. This fire incident has data, including the date/time, the address, the city, the state, the people involved (which includes homeowners, victims and fire department/police), injuries (which will relate to the people involved), and the monetary damage done to the place set on fire. You gather these facts and **normally write a story**.

And the data, which can be structured into something more useful, just dies in a story. That's it. Well, this app will change that. It'll catalog the data mentioned in the previous graph. It may not be the most scientific thing, but these are the types of projects that every news organization that covers a city could build. And thus you shall build it, o wonderful hacker-journo pioneer.

## WELL HOW DO I DO THAT

First we need to set up the models.py file. That's where your data architecture lives. If you've done any work with SQL, some of this may seem familiar. Let's begin.

First, download some sort of text editor. **Here's a link** to a few good free ones. I prefer **Coda**, of course, but to each their own. What you code in doesn't really matter, although we pretend like it is. After you've downloaded it, start a new file and name it models.py. Open it up. This is where we will type our code.

Now, with Django, you have something called a class. In real simple terms, think of each class as an individual spreadsheet page. It has fields, rows and connections to other spreadsheets (classes). So first we need to build the location portion. A location has a street address, a city and a zip code. A city has a state. So we need to build an individual class for state, zip code, city and then location. It looks like this:

```

view plain copy to clipboard print ?
1. from django.db import models
2.
3. class State(models.Model):
4.     name = models.CharField(max_length=50)
5.     name_slug = models.SlugField()
6.     short_name = models.CharField(max_length=8)
7.     def __unicode__(self):
8.         return self.name
9.
10. class City(models.Model):
11.     name = models.CharField(max_length=150)
12.     name_slug = models.SlugField()
13.     state = models.ForeignKey(State)
14.     def __unicode__(self):
15.         return self.name

```

Now let me explain to you how this works. Each of these are classes, which are tables within the big FireTracker database. A state has a field name, a field name\_slug, and a short name. A city has a name, its name\_slug and a state. Because the city NEEDS a state, you need to make that class first. If the City needed a color field, for some instance, you would have also made that class BEFORE the City class.

The name\_slug is used for urls, and the def \_\_unicode\_\_ business is so you can spit out the name in the admin and other spots quicker. Trust me, it'll all make sense later.

Well, what other classes do we need? Well, we need fire departments, people, the location of the fire, an all-encompassing fire class, and others. Here's the entire thing, and if it doesn't make sense right now, that's okay, just take a gander at what a functional app's models.py file looks like:

```

view plain copy to clipboard print ?
1. from django.db import models
2.
3. class State(models.Model):
4.     name = models.CharField(max_length=50)
5.     name_slug = models.SlugField()
6.     short_name = models.CharField(max_length=8)
7.     def __unicode__(self):
8.         return self.name

```

```
9.
10. class City(models.Model):
11.     name = models.CharField(max_length=150)
12.     name_slug = models.SlugField()
13.     state = models.ForeignKey(State)
14.     def __unicode__(self):
15.         return self.name
16.
17. class Department(models.Model):
18.     name = models.CharField(max_length=150)
19.     name_slug = models.SlugField()
20.     short_name = models.CharField(max_length=15)
21.     def __unicode__(self):
22.         return self.name
23.
24. class Title(models.Model):
25.     title = models.CharField(max_length=50)
26.     title_short = models.CharField(max_length=10,
27. blank=True, null=True)
28.     title_slug = models.SlugField()
29.     employer = models.ForeignKey(Department, blank=True,
30. null=True)
31.     def __unicode__(self):
32.         return self.title
33.
34. class Person(models.Model):
35.     first_name = models.CharField(max_length=150)
36.     last_name = models.CharField(max_length=150)
37.     name_slug = models.SlugField()
38.     dob = models.DateField(blank=True, null=True)
39.     title = models.ForeignKey(Title, blank=True, null=True)
40.     experience = models.IntegerField(blank=True, null=True)
41.     def __unicode__(self):
42.         return "%s %s" % (self.first_name, self.last_name)
43.     def get_absolute_url(self):
44.         return "/firetracker/person/%s/%s" % (self.id,
45. self.name_slug)
```



```

45.         street = models.CharField(max_length=150)
46.         street_slug = models.SlugField()
47.         city = models.ForeignKey(City)
48.         property_value = models.IntegerField(max_length=12,
49. blank=True, null=True)
50.         owner = models.ManyToManyField(Person, blank=True,
51. null=True)
52.
53.     def __unicode__(self):
54.         return self.street
55.
56.
57. class Station(models.Model):
58.     name = models.CharField(max_length=150)
59.     name_slug = models.SlugField()
60.     department = models.ForeignKey(Department)
61.     address = models.ForeignKey(Address)
62.     def __unicode__(self):
63.         return self.name
64.
65.
66. class StoryLink(models.Model):
67.     link = models.CharField(max_length=250)
68.     headline = models.CharField(max_length=250, null=True,
69. blank=True)
70.     date = models.DateField(null=True, blank=True)
71.     def __unicode__(self):
72.         return self.headline
73.
74.
75. class Injury(models.Model):
76.     injury = models.CharField(max_length=150)
77.     injury_slug = models.SlugField()
78.     def __unicode__(self):
79.         return self.injury
80.
81.
82. class Victim(models.Model):
83.     person = models.ForeignKey(Person)
84.     injury = models.ForeignKey(Injury)
85.     def __unicode__(self):
86.         return "%s %s" % (self.person.last_name,
87. self.injury.injury)
88.
89.
90. class Source(models.Model):

```

```

80. class Source(models.Model):
81.     source = models.ForeignKey(Person)
82.     def __unicode__(self):
83.         return self.source.name_slug
84.
85. class Cause(models.Model):
86.     type = models.CharField(max_length=150)
87.     type_slug = models.SlugField()
88.     def __unicode__(self):
89.         return self.type
90.
91. class Fire(models.Model):
92.     location = models.ForeignKey(Address)
93.     cause = models.ForeignKey(Cause, blank=True, null=True)
94.     date = models.DateTimeField(blank=True, null=True)
95.     monetary_damage = models.IntegerField(blank=True,
96. null=True)
97.     respondings = models.ManyToManyField(Station,
98. blank=True, null=True)
99.     response_time = models.DateTimeField(blank=True,
100. null=True)
101.     extinguish_time = models.DateTimeField(blank=True,
102. null=True)
103.     story_links = models.ManyToManyField(StoryLink,
104. blank=True, null=True)
105.     victims = models.ManyToManyField(Victim, blank=True,
106. null=True)
107.     source = models.ForeignKey(Person, blank=True,
108. null=True)
109.     def __unicode__(self):
110.         return "%s on %s" % (self.location, self.date)
111.     def time_took(self):
112.         return self.response_time - self.date
113.     def extinguish_time_took(self):
114.         return self.extinguish_time - self.response_time
115.     def get_absolute_url(self):
116.         return "/firetracker/fire/%s/%s" % (self.id,
117. self.location.street_slug)

```

Kind of complicated, isn't it? Well, that's the point. Each class contains a field that will eventually be called

later from our database and spit out onto a page. Or it will be used to help make a url. The last three parts — `def time_taken` and `whatnot` — are defining variables that we will use later. Maybe if you look at the words and the math you'll figure out what we're calculating here.

Now, using your text editor, you will copy and paste this code into a file you will name `models.py`. You will then upload it to your `/opt/django-projects/firetracker/` directory.

So boom. Your models are done. Next we will create a database and sync the models to it. And then we will create an admin so you can look at what you've made and enter data in the backend. So **move on to Step Five.**

## Step Five: Setting up your database and `settings.py` file

We've got our `models.py` file **figured out in Step Four**. Next we need to set up a database for `models.py` to talk to, and then edit our `settings.py` file so it knows where everything lives.

First let's set up a database in PostgreSQL via the command line, which **you can also learn at this site**. So, ssh into your server using a terminal app, like **I showed you in Step One**. Then we're going to add a user who has the right permissions to interact with the database. Let's call the user `bobloblaw`. Once you're logged in, you make `bobloblaw` (or whatever) like this:

```
sudo adduser bobloblaw
```

Add a password and then remember it. Now we need to switch over to the `postgres` user, which isn't that hard. Type:

```
sudo su postgres
```

Your command line will now have "`postgres`" in front of it. Next we need to open up the `postgres` command line, which is simple, by typing:

```
psql template1
```

Well, now we need to add bobloblaw by typing:

```
CREATE USER bobloblaw WITH PASSWORD 'lawblog';
```

Remember the semi-colon at the end. It's very important. Let's create our table for firetracker. Type:

```
CREATE DATABASE firetracker;
```

And lastly we must give the database permissions to be used by bobloblaw:

```
GRANT ALL PRIVILEGES ON DATABASE firetracker to bobloblaw;
```

Woohoo! We did it! Now type \q to quit.

## SETTING UP SETTINGS.PY

Now connect with your FTP client and go to /opt/django-projects/firetracker and open settings.py. This is sort of where you tell your app which database you're using, where it lives, etc. Here's how you should have it set up, but first read the code and then I'll tell you about it:

```
view plain copy to clipboard print ?
1. # Django settings for firetracker project.
2.
3. DEBUG = True
4. TEMPLATE_DEBUG = DEBUG
5.
6. ADMINS = (
7.     ('Your Name', 'yourname@email.com'),
8. )
9.
10. MANAGERS = ADMINS
11.
12. DATABASES = {
```

```

12. DATABASES = {
13.     'default': {
14.         'ENGINE': 'postgresql_psycopg2', # Add
        'postgresql_psycopg2', 'postgresql', 'mysql', 'sqlite3' or
        'oracle'.
15.         'NAME': 'firetracker', # Or
        path to database file if using sqlite3.
16.         'USER': 'bobloblaw', # Not used
        with sqlite3.
17.         'PASSWORD': '', # Not used with
        sqlite3.
18.         'HOST': '', # Set to empty
        string for localhost. Not used with sqlite3.
19.         'PORT': '', # Set to empty
        string for default. Not used with sqlite3.
20.     }
21. }
22.
23. # Local time zone for this installation. Choices can be
    found here:
24. # http://en.wikipedia.org/wiki/List\_of\_tz\_zones\_by\_name
25. TIME_ZONE = 'America/New_York'
26.
27. # Language code for this installation. All choices can be
    found here:
28. # http://www.i18nguy.com/unicode/language-identifiers.html
29. LANGUAGE_CODE = 'en-us'
30.
31. SITE_ID = 1
32.
33. # If you set this to False, Django will make some
    optimizations so as not
34. # to load the internationalization machinery.
35. USE_I18N = True
36.
37. # If you set this to False, Django will not format dates,
    numbers and
38. # calendars according to the current locale
39. USE_L10N = True
40.
41. # Absolute filesystem path to the directory that will hold
    user-uploaded files.

```

```

42. # Example: "/home/media/media.lawrence.com/"
43. MEDIA_ROOT = ''
44.
45. # URL that handles the media served from MEDIA_ROOT. Make
    # sure to use a
46. # trailing slash if there is a path component (optional in
    # other cases).
47. # Examples: "http://media.lawrence.com",
    # "http://example.com/media/"
48. MEDIA_URL = ''
49.
50. # HEY this next line? We will add something here in the next
    # step. But for now, you can just leave in this place holder
51. ADMIN_MEDIA_PREFIX =
    'http://s3.amazonaws.com/bucketname/projectname/admin/'
52.
53. # Make this unique, and don't share it with anybody.
54. # This shouldn't actually say blah because you should use
    # whatever was originally here
55. SECRET_KEY = 'blah'
56.
57. # List of callables that know how to import templates from
    # various sources.
58. TEMPLATE_LOADERS = (
59.     'django.template.loaders.filesystem.Loader',
60.     'django.template.loaders.app_directories.Loader',
61.     # 'django.template.loaders.eggs.Loader',
62. )
63.
64. MIDDLEWARE_CLASSES = (
65.     'django.middleware.common.CommonMiddleware',
66.     'django.contrib.sessions.middleware.SessionMiddleware',
67.     'django.middleware.csrf.CsrfViewMiddleware',
68.     'django.contrib.auth.middleware.AuthenticationMiddleware',
69.     'django.contrib.messages.middleware.MessageMiddleware',
70. )
71.
72. ROOT_URLCONF = 'firetracker.urls'
73.
74. TEMPLATE_DIRS = ("/opt/django-

```

```

    projects/firetracker/templates"
75.     # Put strings here, like "/home/html/django_templates"
    or "C:/www/django/templates".
76.     # Always use forward slashes, even on Windows.
77.     # Don't forget to use absolute paths, not relative
    paths.
78. )
79.
80. INSTALLED_APPS = (
81.     'django.contrib.auth',
82.     'django.contrib.contenttypes',
83.     'django.contrib.sessions',
84.     'django.contrib.sites',
85.     'django.contrib.messages',
86.     'django.contrib.admin',
87.     'firetracker.fires',
88. )
89.
90. # Ignore this next bit, but we will put something here later
    that you will like
91. from S3 import CallingFormat
92. DEFAULT_FILE_STORAGE = 'storages.backends.s3.S3Storage'
93. AWS_ACCESS_KEY_ID = 'your_access_key_id'
94. AWS_SECRET_ACCESS_KEY = 'your_secret_access_key'
95. AWS_STORAGE_BUCKET_NAME = 'andymboyle'
96. AWS_CALLING_FORMAT = CallingFormat.SUBDOMAIN
97. AWS_HEADERS = {
98.     'Expires': 'Sat, 15 Apr 2012 20:00:00 GMT',
99.     'Cache-Control': 'max-age=86400',
100. }
101. AWS_STORAGE_ACL = 'public-read'
102. # This next part, if you use gmail, enter in your email at
    gmail.com and your password
103. # This way we can send error messages to you and that is so
    much fun
104. EMAIL_HOST = 'smtp.gmail.com'
105. EMAIL_HOST_USER = 'youremail@email.com'
106. EMAIL_HOST_PASSWORD = 'yourpasswordgoeshere'
107. EMAIL_PORT = 587
108. EMAIL_USE_TLS = True

```

## WHAT THE HELL IS THIS SHIT

So a lot of this stuff may be over your head. Don't worry, because it slowly makes sense once you see it in action. Most of it you can just leave alone, but there's a few spots that you can edit now or will have to edit later. This code is how yours should sort of look, with you filling the blanks in later. The `Debug = True` bit is just so you can see errors when they happen. The `ADMINS` part should make sense — you want to be able to login to the admin with a name and whatnot. So put in your name and email.

The `databases` bit is the important part: You're telling it to use `postgresql` and `psycopg2`, which is a fancy way of saying "Use this database backend, please." Then you're giving it the name of the database — remember that up there? — and the user that can access the database on the server.

Scroll down to `SECRET_KEY`. I threw `blah` in there as placeholder. Your current `settings.py` file has a long random string here, so you'll want to use that instead of `blah`, obviously. `TEMPLATE_DIRS` is where we're going to throw the template folder, which I will explain a later post. `ADMIN_MEDIA_PREFIX` I will teach you about in my next post.

Under `INSTALLED_APPS` we add the app we just created. See it there? `'firetracker.fires'`, is the code we add. We also added `'django.contrib.admin'`, as well, which I will explain in the next post.

The `from S3 import CallingFormat` won't be used in this tutorial, but I thought you should have it. You use this if you want to add something to upload media, perhaps photos of people or fires. Who knows. But I wanted to leave it in there in case I expand this later.

And lastly, the part that begins with `EMAIL_HOST` will allow you to add an email client. This one is set up for gmail. I would highly suggest if you're creating an app you make some sort of gmail account that's called `appnameisbroke@gmail.com` and then forward it to yourself and any other developers on your team. This way, once a project is in production, you'll get emails when stuff breaks. This is important.

So, you can copy the code from up there, or [copy the code from here](#) and put it into your `settings.py` file via your FTP client and text editor.

## SYNCDDB TIME HELLS YEAH

Now comes the awesomest part. Go into your `/opt/django-projects/firetracker` directory and type this:



```
python manage.py syncdb
```

If nothing's broken, it'll ask you to create a superuser. This will be a username to log into the admin later. Remember it. And then if a bunch of stuff runs across the screen and ends with something like "No fixtures found." then you did it right! Go grab a beer and praise **Jacob Kaplan-Moss** because your goddamn syncdb worked!

So what did we do? We now made it so your app can talk with the database backend. Because we're not going to be making any changes to the models.py file, we don't have to worry about much. But if we were, **you'd want to install South**, which is a totally rad thing to install and allows you to make changes to your models without having to go into the database and doing SQL queries to drop tables and yatta yatta yatta. Just use South if you plan on editing this app.

**Next we shall set up the views and some basic templates.** I know, you're excited and probably can't wait, either.

## Step Six: The Views and Templates

**Now that the database is set up**, it's time to make our views and templates work. In essence, the views talk to the models, or the database. The templates then talk to the views and spit out the content. It's super awesome.

So, let's edit our views.py file. That will be located in `/opt/django-projects/firetracker/fires/`

Here's the code to copy and paste into the views.py and then save:

```

view plain copy to clipboard print ?
1.  from firetracker.fires.models import State, City, Address,
    Department, Station, Title, Person, StoryLink, Injury,
    Victim, Fire, Cause
2.
3.  from django.shortcuts import render_to_response, redirect,
    get_list_or_404, get_object_or_404
4.  import urllib
5.  from django.conf import settings
6.
7.  def index(request):
8.      fires = Fire.objects.all().order_by('date')[:5]
9.      return render_to_response('index.html', {
10.         'fires': fires,
11.     })
12.
13.  def fire(request, address, un_id):
14.      fire = get_object_or_404(Fire,
15.         location__street_slug=address, id=un_id)
16.      return render_to_response('fire_detail.html', {
17.         'fire': fire,
18.     })
19.
20.  def person(request, slug, un_id):
21.      person = get_object_or_404(Person, name_slug=slug,
22.         id=un_id)
23.      return render_to_response('person_detail.html', {
24.         'person': person,
25.     })

```

The first line brings in information from your models.py file, pulling in all of the classes we plan on using. The next few lines are to say “hey this is a views.py file!”

Line 7 is where we set up up our first page. This is the index, which will be our main /firetracker/ page in the URL. We will set up the URLs later in this post. But for now, let me explain some of the basics.

Line 8 makes a variable called “fires,” and we’re saying that fires should find every “Fire” in the database (Fire is a class we made in the models.py file, remember?). The .objects.all() part is saying “Find everything and spit it out.” The .order\_by('date') is telling it to list everything by date, which is something

each Fire contains. Then the `[:5]` means “Show five.” If you wanted to show 15, it’d be `[:15]`. Then we tell it on line 9 to bring the information and spit it onto a template called `index.html`, which we will create later in this post.

Lastly, on line 10 we say that to allow the variable “fires” to be the same as the “fires” we defined on line 8. This means “fires” can be something we will call on the `index.html` page. Trust me, this will make sense later.

The next page, starting at line 13, is more complicated. This is for each individual fire page. You may notice that we define fire — `def fire` — and then ask for `(request, address, un_id)`. So the request we can ignore. But the address and `un_id` are also variables from the `urls.py` file (this will all make more sense later). Line 14 is similar to line 8, except we’re doing a few extra things.

First, we tell it to `get_object_or_404`. This means if it doesn’t find anything matching in the database, show the `404.html` page. The bits in the parenthesis — `(Fire, location__street_slug=address, id=un_id)` — is trying to make a match. It’s saying, “Okay, look in the Fire class. See if you can find something that has a location slug that is equal to the ‘address’ part of the url. Also, the Fire’s unique ID has to match the ‘id’ part of the url.”

For instance, the url will look something like `/firetracker/fire/2/500-e-10th-street/` — 2 is the unique ID of each fire, and 500-e-10th-street is a web-made version of 500 E. 10th Street. In the admin file we create later, we’ll make something that automatically makes slug files, so don’t worry about this right now.

And then on line 19, we are making the individual person page. Similar as the individual fire page, but we’re matching the name and each person’s unique ID.

## EDITING THE URLS.PY FILE

Now we gotta edit the `urls.py` file, which is located in the same directory as your views — `/opt/django-projects/firetracker/fires/` — so open that up in your FTP client.

**Go to this link** and copy and paste the code (this isn’t posting into WordPress correctly, FYI). This is where we are defining our url structure. The index page will show at `123.123.123.123/firetracker`, for instance (where `123.123.123.123` is the IP address of your AWS instance).

Line 7 has a bit that says `firetracker.fires.views.index`. That’s telling it “Hey, go to the views file for fires, then find the bit defined as ‘index.’ Use that for this url.” Line 8 and 9 are the more fun bits, as they are the ones that are more interacting with the views. If you see, there’s a `un_id` and address part in line 8. That’s corresponding to lines 13 and 14 in the `views.py` file, where it’s trying to make the URL by first finding if that “fire” exists, and then it’s spitting it out.

I'm sucking at explaining this, so, I apologize. But trust me. Shit will work. You will be happy.

## MAKING THE TEMPLATES

Now that we've got the views.py and the urls.py made, time for some templates. As you may recall from the views.py, we have three templates: the index, the individual fire page and the individual person page. Using your FTP program, make a new folder called "templates" in your /opt/django-projects/firetracker/fires/ directory.

Inside of that, create three new files. Can you guess what they are based on the views.py?

YOU'RE RIGHT. You rule. They are, indeed, fire\_detail.html, index.html and person\_detail.html. You're a fantastic human being, and I'm glad to know you.

Go to each of the following links and then copy and paste them into each of their corresponding files.

[index.html](#)

[person\\_detail.html](#)

[fire\\_detail.html](#)

Awesome. We now have our views. They're not the prettiest ever, but they get the basic point across. **Next we will configure the server to display this stuff.**

## Step Seven: Configuring uWSGI and nginx

**Now that we've set up the views and whatnot**, it's time to make the damn server work. This is probably the part that is the most difficult, in my opinion, as it's the difference between a project being an idea on your laptop and being something that's live and in the world.

After you read this, **it may help to watch this presentation by Jacob Kaplan-Moss**, one of the original creators of Django and an all-around coding badass. It explains deployment much better than I could, and while it's super duper technical, it does go through some of what we're doing here. It can't hurt to learn more, right?

Anywho, if you remember in **Step Two**, we added a few directories, such as /opt/run/ and others. We're using some of the basics that my pal **Jeremy Bowers** set up in **his blog post on uWSGI and nginx**.

What's **nginx** and **uWSGI**, you ask? Nginx is, if you remember from my analogy in **Step One**, akin to the

bartender getting you a beer. UWSGI would be more like if the bartender had to understand German in order to find the right beer. It acts sort of as a translator between your app and the server software.

Now, use your terminal and public key and ssh in to your instance. We will now get and install nginx and uWSGI. Do each line one at a time.

```
cd /opt/  
wget http://projects.unbit.it/downloads/uwsgi-0.9.6.4.tar.gz  
wget http://nginx.org/download/nginx-0.8.51.tar.gz  
tar -xzf nginx-0.8.51.tar.gz  
tar -xzf uwsgi-0.9.6.4.tar.gz
```

Woohoo. We have downloaded stuff. Aren't we fancy? (Yes.) Now you need to compile and install uWSGI and nginx. It may look scary, but don't worry. Enter each line one at a time, as per usual:

```
cd uwsgi-0.9.6.4  
make  
cp uwsgi /usr/sbin
```

Not that hard, right? Now this bit's a little harder. But don't worry, you can just copy and paste stuff and it'll all work out:

```
cd ../nginx-0.8.51/  
touch /opt/run/nginx.pid  
touch /opt/lock/nginx.lock  
touch /opt/log/nginx/error.log
```

Now copy these lines and paste them into the terminal then hit enter:

```
./configure --conf-path=/etc/nginx/nginx.conf --error-log-  
path=/opt/log/nginx/error.log --pid-path=/opt/run/nginx.pid --lock-  
path=/opt/lock/nginx.lock --sbin-path=/usr/sbin\ --without-http-cache
```

Now:

```
make
make install
```

What did we just do? We told nginx where to look for its configuration file, we told it where to put its log file, and a bunch of other wonderfulness. Next we need to do some more fun configuration.

## SETTING UP UWSGI

Go into your project directory, which should be at `/opt/django-projects/firetracker/` and then we will make a few files:

```
mkdir uwsgi
touch uwsgi/__init__.py
touch uwsgi/wsgi_app.py
```

This is where Mr. Bowers would have you edit code in the terminal. But guess what? That always screws me up. So we're just going to use FTP to edit that `wsgi_app.py` file. Go to the `/opt/django-projects/firetracker/uwsgi/` directory and open the `wsgi_app.py` file using whatever FTP thing I told you to use (or whatever you're using), and then paste this inside:

```
#!/usr/bin/env python
import sys, os, django.core.handlers.wsgi
sys.path.append('/opt/django-projects/')
os.environ['DJANGO_SETTINGS_MODULE'] = 'firetracker.settings'
application = django.core.handlers.wsgi.WSGIHandler()
```

Save that bad boy. Now we're going to make a configuration file for this project that'll help you control uWSGI. Do this:

```
cd /etc/init/
touch firetracker.conf
```

This file will allow you to quickly restart your app, using a simple command like `“sudo service firetracker`

restart.” You’ll need to do that in the future if you ever change a view, model or admin file. So it’s good to have a shortcut. Now open up the file at `/etc/init/firetracker.conf` and throw this information inside:

```
view plain copy to clipboard print ?  
1. description "uWSGI server for ProjectFiretracker"  
2. start on runlevel [2345]stop on runlevel [!2345]  
3. respawnexec /usr/sbin/uwsgi --socket  
   /opt/run/firetracker.sock --chmod-socket --module wsgi_app -  
   -pythonpath /opt/django-projects/firetracker/uwsgi -p 8
```

Jeremy’s post explains more about what this means. But hey, this is just meant to get you started. You can learn more about how the hell this all works later. Next thing you need to do is run this command to let the server know it has a new toy:

```
sudo initctl reload-configuration
```

And then do this to start your uWSGI instance:

```
sudo service firetracker start
```

You can also replace start with stop, or restart, and it’ll do what you think it should do. Just remember that any change you make to most of the core Django files, you’ll need to restart firetracker by ‘sudo service firetracker restart’ so it pops in the changes. Also remember that this needs to be running as well as nginx or else nothing will show up. Speaking of which, let’s set that up.

## SETTING UP NGINX

*NOTE: Some pals have had errors with the nginx.conf later described in this tutorial. If you do, e-mail me at my website address at the email service Google happens to use.*

The last thing we need to do is edit our nginx configuration file so it can talk to the uWSGI instance you just created. Using FTP, open up this file `/etc/nginx/nginx.conf`. Copy and paste this code inside and then save:

[view plain](#) [copy to clipboard](#) [print](#) ?

```
1.  user www-data;
2.  worker_processes 1;
3.  error_log /opt/log/nginx.log;
4.  pid /opt/run/nginx.pid;
5.  events {
6.  worker_connections 1024;
7.  use epoll;
8.  }
9.  http {
10.
11.  include /etc/nginx/mime.types;
12.  default_type application/octet-stream;
13.  access_log /var/log/nginx/access.log;
14.  keepalive_timeout 65;
15.  proxy_read_timeout 200;
16.  sendfile on;
17.  tcp_nopush on;
18.  tcp_nodelay on;
19.  gzip on;
20.  gzip_min_length 1000;
21.  gzip_proxied any;
22.  gzip_types text/plain text/css text/xml
23.  application/x-javascript application/xml
24.  application/atom+xml text/javascript;
25.  proxy_next_upstream error;
26.
27.  server {
28.  listen 80;
29.  server_name your-long-ec2-instance-name-goes-here
30.  123.123.123.123;
31.  client_max_body_size 50M;
32.  root /var/www/ht2;
33.  location / {
34.  uwsgi_pass unix://opt/run/firestarter.sock;
35.  include uwsgi_params;
36.  }
```



The only bit you should really notice is the the line that starts with `server_name`. After that you need to put your long EC2 name, which you can find in your AWS administrative panel on your EC2 page. Click on your running instance and then in the bottom window that pops up, look for “Public DNS.” This should look like `ec2-123-123-123-123.compute-1.amazonaws.com` or something. Copy that and paste it into the area after server name. And then take the numbers in the `123-123-123-123` — which is your IP address — and paste them after the instance public DNS (with a space in between).

Now run this:

```
sudo service nginx start
```

And voila! You have uWSGI and nginx running, listening on port 80. You won’t need to “`sudo service nginx restart`” unless you change something in the `.conf` file, which you won’t need to do unless you add another project in the future. Don’t worry about that for now.

Go to `123.123.123.123/firetracker/` It should spit out some basic information. But nothing will load because you have no data entered! We will do that in the next post. **Now let’s move on to Step Eight, which sets up the admin.**

## Step Eight: Configuring the admin and uploading to S3

**We set up our uWSGI and nginx conf files** and our server is now running. Now we will set up the admin to add content — one of the totally sweet parts of Django — and then show you how to get your admin files onto Amazon S3.

What sets Django apart from many other out-of-the-box frameworks is that it has a pretty bitchin’ admin built into it. You don’t have to use it, of course. You could just do everything from your terminal, and add information in through the shell. But Django was designed by developers who worked for a newsroom. Thus bits of it had to be understood by regular newsroom folk — editors, reporters, photographers, etc. People who weren’t super web-savvy should be able to use it through some sort of web interface.

Hence setting up the admin. When you’re done, it’ll look like this:

## Django administration

### Site administration

Auth		
Groups	<a href="#">+ Add</a>	<a href="#">Change</a>
Users	<a href="#">+ Add</a>	<a href="#">Change</a>
Fires		
Addresss	<a href="#">+ Add</a>	<a href="#">Change</a>
Causes	<a href="#">+ Add</a>	<a href="#">Change</a>
Citys	<a href="#">+ Add</a>	<a href="#">Change</a>
Departments	<a href="#">+ Add</a>	<a href="#">Change</a>
Fires	<a href="#">+ Add</a>	<a href="#">Change</a>
Injurys	<a href="#">+ Add</a>	<a href="#">Change</a>
Persons	<a href="#">+ Add</a>	<a href="#">Change</a>
States	<a href="#">+ Add</a>	<a href="#">Change</a>
Stations	<a href="#">+ Add</a>	<a href="#">Change</a>
Story links	<a href="#">+ Add</a>	<a href="#">Change</a>
Titles	<a href="#">+ Add</a>	<a href="#">Change</a>
Victims	<a href="#">+ Add</a>	<a href="#">Change</a>
Sites		
Sites	<a href="#">+ Add</a>	<a href="#">Change</a>

Neat, right? This looks like something your average person who's ever added a photo on Facebook should be able to work, right? Right.

So, first you need to add a file in your project directory (/opt/django-projects/fires/). Name it admin.py. You can do this by either creating a file on your computer and then uploading it, or if you're using something like Coda, just right click on the in the area listing files and click New File.

Copy and paste this code inside. I'll explain afterword how it works:

```
view plain copy to clipboard print ?  
1. from django.contrib import admin  
2. from firetracker.fires.models import State, City, Address,  
   Department, Station, Title, Person, StoryLink, Injury,  
   Victim, Fire, Cause  
-
```

```
3.
4. class StateAdmin(admin.ModelAdmin):
5.     prepopulated_fields = {'name_slug': ('name', ) }
6.     search_fields = ['name']
7.
8. class CityAdmin(admin.ModelAdmin):
9.     prepopulated_fields = {'name_slug': ('name', ) }
10.    search_fields = ['name']
11.
12. class AddressAdmin(admin.ModelAdmin):
13.     prepopulated_fields = {'street_slug': ('street', ) }
14.     search_fields = ['street']
15.
16. class DepartmentAdmin(admin.ModelAdmin):
17.     prepopulated_fields = {'name_slug': ('name', ) }
18.     search_fields = ['name']
19.
20. class StationAdmin(admin.ModelAdmin):
21.     prepopulated_fields = {'name_slug': ('name', ) }
22.     search_fields = ['name']
23.
24. class TitleAdmin(admin.ModelAdmin):
25.     prepopulated_fields = {'title_slug': ('title', ) }
26.     search_fields = ['title']
27.
28. class PersonAdmin(admin.ModelAdmin):
29.     prepopulated_fields = {'name_slug': ('first_name',
30.     'last_name' ) }
31.     search_fields = ['name']
32.
33. class InjuryAdmin(admin.ModelAdmin):
34.     prepopulated_fields = {'injury_slug': ('injury', ) }
35.     search_fields = ['injury']
36.
37. class CauseAdmin(admin.ModelAdmin):
38.     prepopulated_fields = {'type_slug': ('type', ) }
39.     search_fields = ['type']
40.
admin.site.register(State, StateAdmin)
```

```
41. admin.site.register(City, CityAdmin)
42. admin.site.register(Address, AddressAdmin)
43. admin.site.register(Department, DepartmentAdmin)
44. admin.site.register(Station, StationAdmin)
45. admin.site.register>Title, TitleAdmin)
46. admin.site.register(Person, PersonAdmin)
47. admin.site.register(StoryLink)
48. admin.site.register(Injury, InjuryAdmin)
49. admin.site.register(Victim)
50. admin.site.register(Cause, CauseAdmin)
51. admin.site.register(Fire)
```

## WHAT IS THIS SHIT?

So, if you look at this and compare it with your models.py file, you'll see a lot of things overlapping. First, you need to pull in everything from your models.py file. This line tells Django to import its admin bits:

```
from django.contrib import admin
```

The next line pulls in the various classes from your models.py file. This allows you to later use them in this admin.py file.

Line 4-6 is the basic way we're setting up everything in the admin. So you need to make a class for the State class. Thus StateAdmin. Line 5 is telling you to pre-populate the name\_slug field (which you have in your models.py file, remember?) with whatever is typed in the name slug. You'll see this in action once everything is live. Then search\_fields allows you to include a field to search by in the admin.

Line 40 is where you basically sync stuff up. You're sort of saying "Hey, State and StateAdmin, make friends!" So you need one of these for each of the previous admin classes you created above.

## UPLOADING SHIT TO S3

Now we need to upload some files to S3. Why, do you ask? Because the admin uses some CSS, Javascript and styling deals. Well, why don't we just put that on the instance we created and are hosting

the Django stuff on? Because that would require us to also set up a server to run media files. That would mean more hits to your server, more traffic, more load, etc. And as media files in general — photos, audio, video, etc. — can be large, that means it would also push up your Amazon bill, as they charge for traffic leaving an instance after 1 gigabyte.

Also, I don't know how to set up a media server that isn't for PHP. SO THERE.

First, we are going to download a copy of Django onto our harddrive and then pull out the media bits. **Go here to download** a copy. Make sure it's your current version — if you have Django 1.2.3 running, get that. If it's 1.3, download that version. After it's downloaded, unzip it and look inside. Go to the unzipped file, then this path `/django/contrib/admin/media/`

Inside of this are all the files you're going to want to upload to your S3 account. This is sort of an annoying process, and I'm sure there's a quicker way, but I'm kind of dumb so I end up doing things the longer way. (AKA hey anyone out there reading this, if you have a better way to do this, leave a comment or let me know, and I'll include it in here and give you props).

So in the end, you'll want to have uploaded to S3 a css, img and js folder, with all the files in each individual folder, and then each individual folder will have its subsequent subfolders and files. Like I said, this will be annoying and tedious.

To get this set up, login with the account you created earlier to AWS. Click on S3 once you're logged in. You will now create a bucket. I suggest you either name it after your organization — `dailynewstribuneglobecompany` — or something. Or name it after your project. Whatever. Once you do that, you need to create a folder named "admin." Inside of that, add three more folders named css, img and js. And from there, you will upload each individual file, create new subfolders, etc., so it matches the file on your computer. Do that shit like I said you should do up there.

Once that's done, go back to the main bucket directory. Right click on the admin folder, and then click Make Public. This will make it so people can look at your files without the various AWS credentials.

Now, to test that that worked, go to the equivalent URL that looks like this, only instead of `dailynewstribuneglobecompany` it's whatever you threw up there:  
`https://s3.amazonaws.com/dailynewstribuneglobecompany/admin/img/admin/arrow-down.gif`

You should be able to see a little arrow appear in the screen of your top left browser. If you can see that, voila, your admin shit was uploaded.

Once that's done, you will need to edit your settings.py file, like we did in Step Five.

Change line 51 that to:

```
ADMIN_MEDIA_PREFIX = 'http://s3.amazonaws.com/dailynewstribuneglobecompany/admin/'
```

This tells Django to look for the admin stylesheets here. Now that your project is live at 123.123.123.123/firestarter/, go there and then add /admin/ to the end of the URL. You can now add in the username and password you entered when you originally syncdb'd back in Step Five.

Once you're logged in, you should see the basic admin screen that we showed earlier. In my next post, I'll discuss adding content.