

# From 0 to DIY Web App in 120 minutes

Daniel Axtens  
daniel@axtens.net  
@daxtens

September 30, 2013

# Outline

## Getting Started

Anatomy of a simple web app

Overview

Delivery

## Building your app

Server

Database

Front end

## Deploying your app!

## Further directions

## When you should not use this

- ▶ Static web site → static HTML and CSS.
- ▶ User-editable content — blog/corporate website → Content Management System
- ▶ Data driven and highly structured data, well understood system based around Create Retrieve Update Delete (CRUD) → e.g. Django

## When you should use this

- ▶ Something requiring persistent storage but where the data is not highly structured, or has a frequently changing schema or is otherwise difficult to reduce to a relational database → a non-relational database (NoSQL): e.g. MongoDB
- ▶ Something so small that setting up something like Django will take longer than actually implementing the functionality → a micro-framework, e.g. Bottle

## Start thinking...

- ▶ There's no set end product for this workshop. There will be several spots where you can decide for yourself what to do, and get assistance in making it happen.
- ▶ Start thinking about a (relatively) simple application that (preferably) fits the previous criteria, or at least is dynamic with persistent storage. Perhaps:
  - ▶ Collate information from a variety of internet-connected sensors, process it, and display a result. (e.g. average/min/max?)
  - ▶ A database of quotable quotes, with up/down-voting
  - ▶ Configurable Insult/3 word slogan/etc generator
  - ▶ Your idea here.
  - ▶ Really stuck? (Yet another) blog with comments, photo-sharing service, todo list, calendar, social-bookmarking service, etc. (Most of these things are better built with e.g. Django)

## Local environment

- ▶ Set up git to point this to your repository, not mine.
  - ▶ Fork the repo on GitHub.
  - ▶ `git remote set-url origin https://github.com/YOUR USERNAME/diy-web-app`
  - ▶ (Unless you already have private keys set up, in which case use `git@github.com:YOUR USERNAME/diy-web-app.git`)
- ▶ `vagrant up`
- ▶ `vagrant ssh`
- ▶ Any changes you make to your local file system are reflected in the virtual machine in the `/vagrant` directory.

## Remote environment

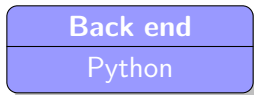
- ▶ You will need to supply your own EC2 or equivalent instance to deploy this. (Or any VPS, really.)
- ▶ AWS (Amazon Web Services) → a set of services.
  - ▶ EC2 - Elastic Cloud Compute → Virtual Machines
  - ▶ SES - Simple Email Service
  - ▶ Route 53 - DNS service → maps names like google.com to IP addresses like 203.8.182.219
  - ▶ S3 - Simple Storage Service → Way to store static assets/files.
  - ▶ CloudFront - CDN (Content Delivery Network) → caches static assets near the user for faster speed, less load on your server.
  - ▶ And many, many more.
- ▶ Not free in general, but a free tier exists.

## Remote Environment

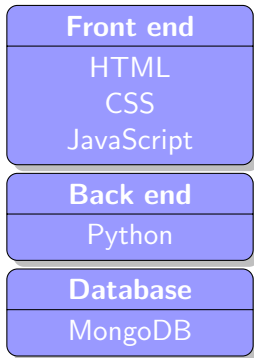
- ▶ Start and log in to an Ubuntu EC2 instance.
- ▶ Update the package lists and packages: `sudo apt-get update; sudo apt-get -y upgrade; sudo reboot`
- ▶ (All text **like this** is a command you should execute on your EC2 instance)
- ▶ SSH in again after it reboots.



## Anatomy of a simple web app: overview

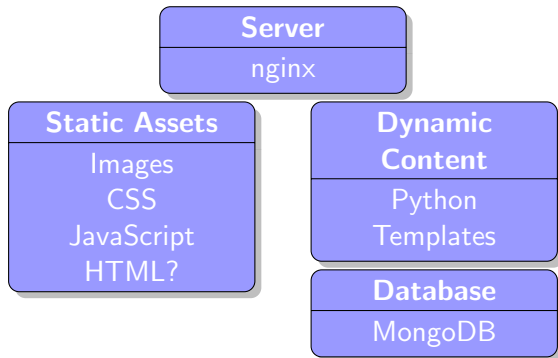


## Anatomy of a simple web app: now add a database



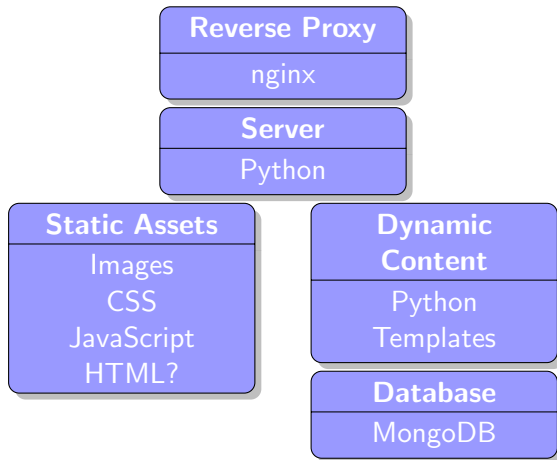
# Anatomy of a simple web app: how does this get to you?

Ideal world



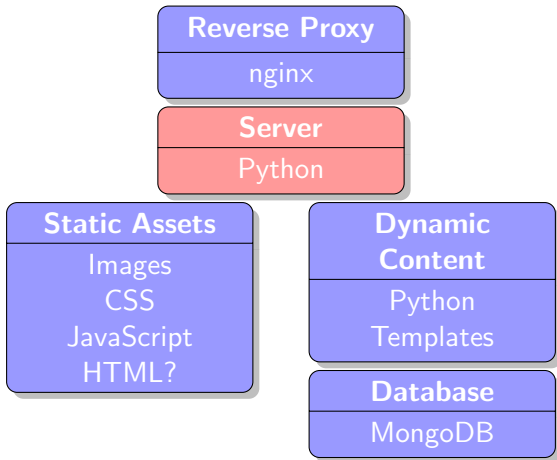
# Anatomy of a simple web app: how does this get to you?

Our world



# Building your app

# Server



# Introducing Bottle



**Figure :** Bottle is a fast, simple and lightweight WSGI micro web-framework for Python. <http://bottlepy.org/>

You might also want to consider Flask:  
<http://flask.pocoo.org/>. It does a bit more for you/has a bit more magic than Bottle.

# Hello World!

Edit webapp.py.

---

```
from bottle import route, run
```

```
@route('/')
```

```
def index():
```

```
    return "Hello World!"
```

```
if __name__ == '__main__':
```

```
    run(debug=True, reloader=True, host='0.0.0.0')
```

---

In your vagrant shell, `cd /vagrant; python webapp.py`.

Then visit `http://localhost:8080` in your browser.



## Best practices: commit your changes!

```
git add webapp.py
git commit -m"Hello world example"
git push
```

## Greeting others

Let's extend:

```
@route('/greet/<name>')  
def greet(name):  
    return "Hello, %s!" % name
```

---

```
git add webapp.py  
git commit -m"Add greet function"  
git push
```

# Danger, Will Robinson!

Spot the subtle error and potential security hole.

## Danger, Will Robinson!

What happens if you go to `localhost:8080/greet/<b>Daniel?`

**This unescaped HTML has the potential (in other circumstances) to cause XSS vulnerabilities.**

Fortunately, it's easy to avoid.

## Safe user input the right way

Don't try to sanitise it yourself.  
You will probably miss an edge case.

Use templates.

```
from bottle import route, run, template

...

def greet(name):
    return template("Hello, {{name}}!", name=name)
```

## Separating presentation and content

We don't want template code in .py files.

Edit `views/hello.tpl`. Fill it with standard HTML, with `{{name}}` somewhere. (Lazy? See the link to the sample on the notes.)

Then:

```
from bottle import route, run, template, view
```

```
...
```

```
@route('/greet/<name>')
```

```
@view('hello')
```

```
def greet(name):
```

```
    return {'name': name}
```

**Bottle's template language (SimpleTemplate Engine) has more features, which we'll cover later...**

## Simplifying things

Why do we have different functions for `/` and `/greet`, when they do mostly the same thing? **A bottle function can handle multiple routes.**

```
@route('/')  
@route('/greet/<name>')  
@view('hello')  
def greet(name="World"):
```

## Best practices: commit your changes!

```
git add webapp.py views/hello.tpl  
git commit -m"Serve Hello World with a template."  
git push
```



## Serving static files

Put them in a separate sub-directory!

```
from bottle import route, run, template, view, static_file
import os
```

```
root = os.path.dirname(__file__)
static_root = os.path.join(root, "static")
```

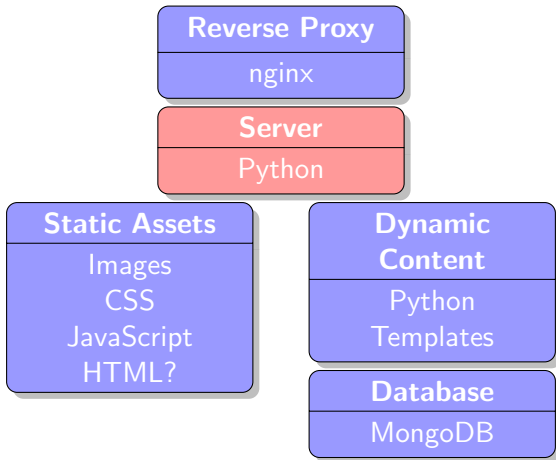
```
...
```

```
@route('/static/<path:path>')
def static(path):
    return static_file(path, root=static_root)
```

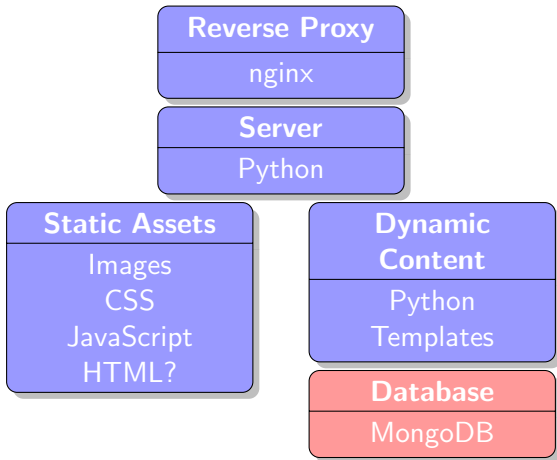
## Best practices: commit your changes!

```
git add webapp.py
git commit -m"Serve static files out of static/"
git push
```

# Server



# Database



# Introducing MongoDB



**Figure :** MongoDB (from “humongous”) is an open-source document database, and the leading NoSQL database.

## NoSQL? Wat? — A diversion on databases

- ▶ Traditional database (e.g. MySQL (→ MariaDB), PostgreSQL, MS-SQL) = Relational Databases
- ▶ “NoSQL” → umbrella term for a large number of non-relational databases.
  - ▶ Document: e.g. MongoDB/CouchDB
  - ▶ Key-value store: e.g. Redis
  - ▶ Other exciting things: e.g. Neo4j - graph database,

Both are *tools for different purposes, with different strenghts and weaknesses*. **Just because MongoDB is cool, doesn't mean it is the solution to all your problems.**

## Why MongoDB for us now?

- ▶ Even with advanced database migration tools like South (for Django), it's still easier to do schema migrations in MongoDB. This helps with the sort of rapid prototyping we're doing here.
- ▶ It's really easy to store files/images in a MongoDB. Not necessarily super-efficient, but again good for prototyping.
- ▶ Often easier to wrap your head around.

# Getting started with MongoDB

- ▶ Key component is a document.
- ▶ Document has a unique ID.
- ▶ Documents are JSON-like.
- ▶ Can contain arbitrary key-value pairs. Values can include arrays and other key-value mappings ('objects')



## An Example

```
{ "title" : "Post",  
  "content" : "I think W, Y, and Z.",  
  "author" : "Daniel Axtens",  
  "date" : ISODate("2013-10-25T08:45:00Z"),  
  "comments" : [{  
    "author" : "Anonymous",  
    "content" : "First P0000st!",  
    "date" : ISODate("2013-09-25T10:01:44.405Z")  
  }, {  
    "author" : "Mr. Helpful",  
    "content" : "I think you might have missed X, \  
                which is listed on my blog.",  
    "link" : "http://helpful.blog.com/",  
    "date" : ISODate("2013-09-25T10:01:44.405Z")  
  }]  
}
```

## Converting this to code

Only chumps and people who enjoy getting owned write their own queries. We use MongoEngine. This is `blog_example.py`:

```
from mongoengine import *
import datetime

# Define the data-structure
class BlogPost(Document):
    title = StringField(required=True)
    content = StringField(required=True)
    author = StringField(required=True)
    date = DateTimeField(required=True, \
                        default=datetime.datetime.now)
    categories = ListField(StringField(max_length=30))
```

Now, let's connect to the database, create and save a post. (This is more of `blog_example.py`)

```
# Connect to the database
```

```
connect('blog_example')
```

```
# Save a post
```

```
post = BlogPost()
```

```
post.title = "Post"
```

```
post.content = "I think W, Y, and Z."
```

```
post.author = "Daniel Axtens"
```

```
post.save()
```

## Exploring your newly created document

```
$ mongo
> use blog_example
> show collections
> db.blog_posts.find()
> db.blog_posts.find({'author': 'Daniel Axtens'})[0]
```

SQL equivalent:

```
SELECT * FROM blog_posts WHERE author = 'Daniel Axtens'
```

## Retriving with MongoEngine

```
# Retrieve a post
posts = BlogPost.objects() # everything
posts = BlogPost.objects(author='Daniel Axtens') # only me
post = posts.first()
print('%s", by "%s": %s' % \
      (post.title, post.author, post.content))

# Update it
post.title = "Freshly Updated Post"
post.save()

# Verify update
print(BlogPost.objects().first().title)
```

## But I want my blog to have comments!

```
class Comment(EmbeddedDocument):
    author = StringField(required=True,
                          default="Anonymous")
    link = URLField()
    content = StringField(required=True)
    date = DateTimeField(required=True,
                          default=datetime.datetime.now)

class BlogPost(Document):
    ...
    comments = ListField(EmbeddedDocumentField(Comment))
```

## But I want my blog to have comments!

```
# Add the comments
```

```
comment = Comment()  
comment.content = "First P0000st!"  
post.comments.append(comment)  
post.save()
```

```
comment = Comment()  
comment.author = "Mr. Helpful"  
comment.content = "I think you've forgotten X..."  
comment.link = "http://helpful.blog.com"  
post.comments.append(comment)  
post.save()
```

## But I want my blog to have comments!

```
# Retrieve a post
posts = BlogPost.objects()
post = posts.first()
print('%s", by "%s": %s' % \
      (post.title, post.author, post.content))
for comment in post.comments:
    if comment.link is not None:
        print('Comment by %s (%s): %s' % \
              (comment.author, comment.link, \
               comment.content))
    else:
        print('Comment by %s: %s' % \
              (comment.author, comment.content))
```



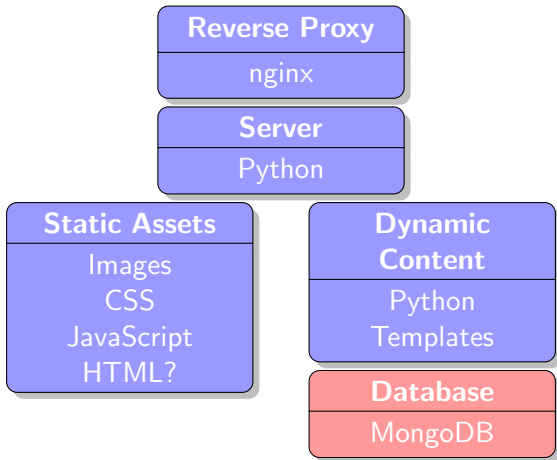
## But I want my blog to have comments!

- ▶ Did you notice the seamless and painless schema migration?
- ▶ Did you notice the really, really painless Object-Document Mapping? Treating a list of comments as a Python list?

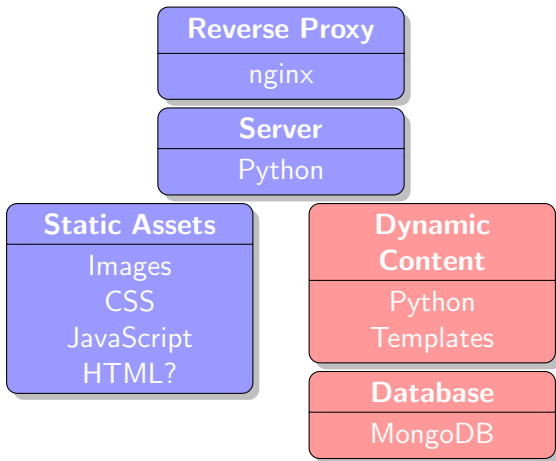
## Over to you...

- ▶ What can a MongoEngine Document contain? (Highlights - see the online documentation for the full list)
  - ▶ BooleanField
  - ▶ DateTimeField
  - ▶ IntField / LongField / DecimalField / FloatField
  - ▶ DictField / MapField
  - ▶ EmailField / URLField
  - ▶ EmbeddedDocumentField
  - ▶ FileField / ImageField
  - ▶ GeoPointField
  - ▶ ListField / SortedListField
  - ▶ ReferenceField
  - ▶ StringField
- ▶ There's also an awesome inheritance system.

# Database



## Serving up your database



## Reading/Writing from your MongoDB

- ▶ I usually put all my class definitions and my `connect(...)` call in a separate file (e.g. `database.py`), then say:

```
from database import *
```

Although there's really no reason you couldn't have it all in one file. (Neatness concerns aside.)

## Simple and stupid example

```
connect('greeter')

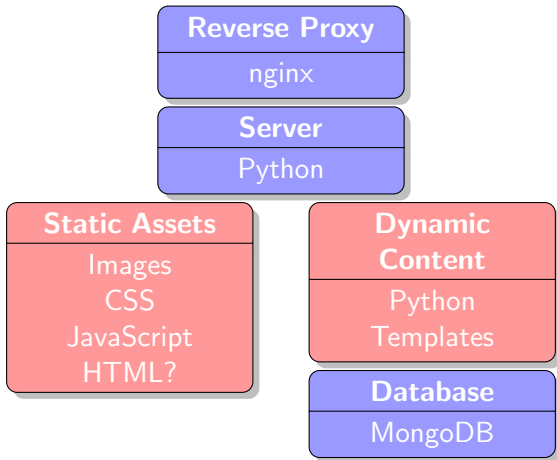
class Name(Document):
    name=StringField(default="World")

@route('/')
@route('/greet/<name>')
@view('hello')
def greet(name=None):
    # todo: not everyone should be able to set.
    dbname = Name.objects().first() or Name()
    if name is not None:
        dbname.name = name
        dbname.save()
    return {'name': dbname.name}
```

Off you go!

Now work on the core of your own app!

## Front end





## Introducing Bootstrap

# Bootstrap

Sleek, intuitive, and powerful mobile first front-end framework for faster and easier web development.

Figure : Bootstrap: [getbootstrap.com](https://getbootstrap.com)

- We're lazy, so we use Bootstrap for the frontend.

# What can I do with Bootstrap?

- ▶ Have things look not-terrible with minimal effort.
- ▶ Have mobile-compatible (“responsive”) layouts basically for free.
- ▶ Get a bunch of integrated client-side behaviours through jQuery.
- ▶ There are more things. (I cannot do design to save my life.)

## Setting up Bootstrap

- ▶ Download it from [getbootstrap.com/getting-started](http://getbootstrap.com/getting-started), extract the archive, put the folders in `static/`
  - ▶ *Don't download the version from the front page. Get the one with the pre-compiled JS and CSS assets.*
- ▶ Choose the example from <http://getbootstrap.com/getting-started/#examples> that best suits your project.
  - ▶ (Optionally) find it on GitHub.
  - ▶ Copy the code to `views/index.tpl`
  - ▶ Download any extra stylesheets, etc.
  - ▶ Adjust relevant links to read `/static/{js, css, etc}/...` (look for `src=`, `href=`)
  - ▶ Fill out the relevant title, meta, etc tags.
- ▶ Adjust your code to use the index template. Include your parameters (e.g. `{{name}}`) in there somewhere :)

## But my submission page/other page/etc is still ugly!

- ▶ You could copy-paste the template code. Or you could save yourself huge amounts of effort by being smart.
- ▶ Create `bootstrapbase.tpl`, and copy-paste the content of `index.tpl`.
- ▶ Delete everything from (and including):

```
<div class="jumbotron">
```

to (and not including):

```
<div class="footer">
```

Replacing it with the text `%include`.

- ▶ Title (replace Slogan Generator with your project name):

```
<title>{{get('title', 'Slogan Generator')}}</title>
```

- ▶ Likewise with meta tags, you can pass them in as a variable if needed.

# But my submission page/other page/etc is still ugly!

- ▶ Now back to `index.tpl`.
- ▶ Delete everything *except* what we deleted from `bootstrapbase.tpl`
- ▶ At the top of `index.tpl` add:  
`%rebase bootstrapbase`

## Contacts page

webapp.py:

```
@route('/contact')
@view('contact')
def contact():
    return {}
```

views/contact.tpl:

```
%rebase bootstrapbase title='Contact'
<p>I have no idea why on earth you'd want to contact me
    over this.<p>
<p>You could <a href="mailto:daniel@axtens.net">email me</a>
    I guess?</p>
```

- ▶ This doesn't deal correctly with the highlighting in the top bar. This is a bit fiddly so I'm not going to go through it now. Some sample code is included, however.
- ▶ As things start to get more complicated (replacing chunks of different pages), it's time to swap out SimpleTemplate for a less simple system (e.g. Jinja). (It's possibly also time to move to a more comprehensive framework.)

## Buzzword time: AJAX

- ▶ AJAX stands for Asynchronous JavaScript and XML.
- ▶ That's not how the term is used any more, however.
  - ▶ JSON (JavaScript Object Notation) >> XML (eXtensible Markup Language)
  - ▶ It can also be synchronous.
  - ▶ Umbrella term for “getting stuff from a server without loading an entire page again”.
- ▶ jQuery (which comes for free with Bootstrap) makes AJAX painless. ... or at least as painless as something in JavaScript can be.



# AJAX: JSON

- ▶ JSON is awesome. We've seen it before with MongoDB. It is also the way JavaScript expresses objects.
- ▶ Key-value, `{...}` for objects, `[...]` for arrays.
- ▶ Trivial to parse and generate in every useful language! (Don't DIY it, and especially don't use JS's `eval` function on it.)
- ▶ Lets make our server spit out JSON!

## Back to webapp.py

```
▶ from bottle import route, run, ..., response
▶ import json
▶ @route('/slogan.json')
  def sloganjson():
      response.set_header('Content-Type',
                          'application/json')
      return json.dumps(randomslogan())
```

## Front end: Introducing jQuery



Figure : jQuery — write less, do more. <http://jquery.com>

- ▶ jQuery makes JavaScript in the browser not suck.
- ▶ I heart it immensely.

## Front end—index.tpl

- ▶ (These will probably be different for you.)
- ▶ Add `id="slogan"` to the `h1` with `{{slogan}}`.
- ▶ Add `id="generate"` to the `a` with `role="button"`. If you've changed the `href`, change it back to `#`.

## Front end—static/js/ajax.js

```
function generateButton() {  
  // get the JSON. TODO: handle errors  
  $.getJSON('/slogan.json', function(data) {  
    // callback when the data is returned  
    // we set the slogan  
    $(slogan).text(data.slogan);  
  });  
}  
  
// when the document is loaded:  
$(document).ready(function(){  
  // wire up the generate button  
  $('#generate').on('click', generateButton);  
});
```

## Including the script—a bit of hackery in `bootstrapbase.tpl`

After the other scripts:

```
<script src="/static/js/ajax.js"></script>
```

Yes, this isn't particularly extensible. But jQuery falls over nicely.

# Hurray, AJAX!

Don't forget to save to git!

## A review: what have we done?

- ▶ A web server written in Python...
- ▶ ... reading and writing data from MongoDB ...
- ▶ ... with a front end made less ugly by Bootstrap ...
- ▶ ... with interactivity through AJAX, implemented in jQuery...
- ▶ ... running in a virtual machine on your computer.



## Deploying your app!

- ▶ As exciting as virtual machines on your computer are...
- ▶ ... virtual machines on someone else's computer are cooler ...
- ▶ ... especially when that other computer is permanently connected to the internet.

## One small thing...

Take out `debug=True`, `reloader=True` from `webapp.py`.

## Re-introducing AWS



**Figure :** Amazon Web Services delivers a set of services that together form a reliable, scalable, and inexpensive computing platform “in the cloud”. `aws.amazon.com`

“Amazon Web Services” and the “Powered by Amazon Web Services” logo, are trademarks of Amazon.com, Inc. or its affiliates in the United States and/or other countries.

## Somewhere to put your code

```
sudo adduser --disabled-password webapp
```

## Getting your source code to the server: introducing Fabric

*Fabric is a Python (2.5 or higher) library and command-line tool for streamlining the use of SSH for application deployment or systems administration tasks.*

<http://fabfile.org>

## fabfile.py

```
from fabric.api import *

def setup():
    sudo("pip install bottle mongoengine bottle-cork")

def deploy():
    # create tarball
    local("tar -czf webapp.tgz webapp.py database.py +  
        " views static")
    # upload and extract
    put("webapp.tgz", "/tmp/webapp.tgz")
    with cd("/home/webapp"):
        sudo("tar -xzf /tmp/webapp.tgz")
    # clean up
    run("rm /tmp/webapp.tgz")
```

## Let's go!

- ▶ `sudo apt-get -y install python-pip python-dev mongodb`
- ▶ `fab -u ubuntu -H number.compcon.dja.id.au setup`
- ▶ `fab -u ubuntu -H number.compcon.dja.id.au deploy`
- ▶ `sudo -u webapp -s python webapp.py`
- ▶ Your code should be running, but the website won't work on port 80, and 8080 is firewalled off.

## What not to do

- ▶ “Why don’t we just set `port=80` in the `run` function call?”



## What not to do

- ▶ “Why don’t we just set `port=80` in the `run` function call?”
- ▶ “Oh, that’s odd, it can’t bind to port 80. Oh, it’s a privileged port? Oh, I can just run it as root!”

## What not to do

- ▶ “Why don’t we just set `port=80` in the `run` function call?”
- ▶ “Oh, that’s odd, it can’t bind to port 80. Oh, it’s a privileged port? Oh, I can just run it as root!”
- ▶ “Yay, it works now!”

## What not to do

- ▶ “Why don’t we just set `port=80` in the `run` function call?”
- ▶ “Oh, that’s odd, it can’t bind to port 80. Oh, it’s a privileged port? Oh, I can just run it as root!”
- ▶ “Yay, it works now!”
- ▶ <ring ring> “Hello, this is developer” ... “What do you mean my app got owned? My server got owned? And wiped out everything else on the server?”

## What not to do

Don't run your app as root.  
Ever. Not even for a little  
while.

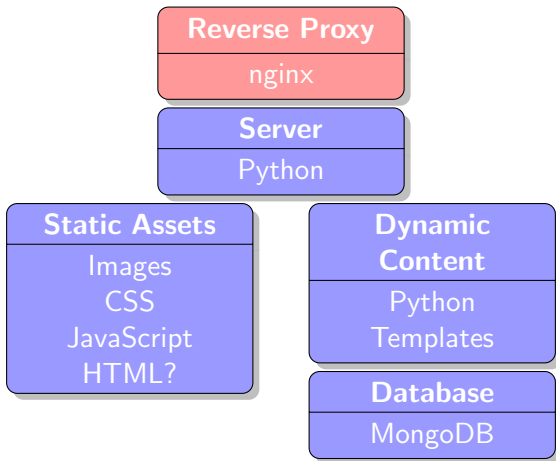
## Introducing our saviour ... nginx



Figure : nginx

```
► sudo apt-get -y install nginx-light
```

# Reverse Proxy



```
sudo nano /etc/nginx/sites-available/default
```

```
server {  
    server_name HOSTNAME_OF_YOUR_SERVER;  
  
    location / {  
        proxy_pass http://localhost:8080/;  
    }  
}
```

- ▶ `sudo service nginx restart`
- ▶ Start your process and (assuming it doesn't rely on a pre-seeded database!) it should work!

## “Staying alive, staying alive” Introducing supervisord



Figure : Supervisord: hang on to your processes.

<http://supervisord.org>

```
► sudo apt-get -y install supervisor
```



```
sudo nano /etc/supervisord/conf.d/webapp.conf
```

```
[program:webapp]
user=webapp
command=python /home/webapp/webapp.py
autostart=true
autorestart=true
```

```
sudo service supervisor stop; sudo service supervisor
start;
```

## Oh no, it's b0rked!

```
► webapp.py:  
  if root:  
    os.chdir(root)
```

## Oh no, it's b0rked!

▶ `webapp.py`:

```
if root:
```

```
    os.chdir(root)
```

▶ `fab -u ubuntu -H number.compcon.dja.id.au deploy`

▶ `sudo supervisorctl restart webapp`

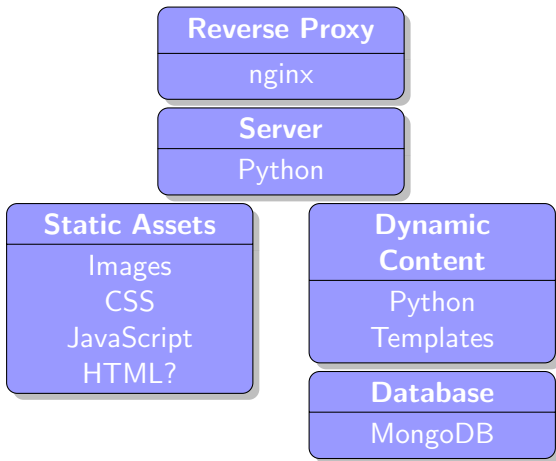
## Oh no, it's b0rked!

- ▶ `webapp.py`:

```
if root:  
    os.chdir(root)
```

- ▶ `fab -u ubuntu -H number.compcon.dja.id.au deploy`
- ▶ `sudo supervisorctl restart webapp`
- ▶ (even better, add the following to `fabfile.py`)  
`sudo("supervisorctl restart webapp")`

# Recap



## Further directions

- ▶ If you're still sorting out bugs, etc., keep working on that.
- ▶ If not, here's some stuff we can look at adding:
  - ▶ Google Analytics
  - ▶ Authentication
  - ▶ Speed improvements
  - ▶ CloudFlare

# Introducing Google Analytics



**Figure :** Google Analytics: Turning data insights into action.  
<http://google.com/analytics>

You can figure this out yourself. Just copy the snippet into `bootstrapbase.tpl`.

# Authentication

- ▶ Security is hard. Don't DIY. Make sure your library seeds the hashes, and uses a hash function designed for passwords (e.g. PBKDF2, not SHA-1 or even worse MD5).



# Authentication

- ▶ Security is hard. Don't DIY. Make sure your library seeds the hashes, and uses a hash function designed for passwords (e.g. PBKDF2, not SHA-1 or even worse MD5).
- ▶ Introducing Cork: <http://cork.firelet.net/>

# Authentication

- ▶ Security is hard. Don't DIY. Make sure your library seeds the hashes, and uses a hash function designed for passwords (e.g. PBKDF2, not SHA-1 or even worse MD5).
- ▶ Introducing Cork: <http://cork.firelet.net/>
- ▶ Seriously consider a more heavy-weight framework.

## “You are not done yet”

- ▶ Your site is slow. `nginx` can make it faster with gzipping, caching and by retrieving assets without going through Bottle.
- ▶ Bottle is slow because it uses a reference WSGI server. Try CherryPy or Paste.
- ▶ Consider CloudFlare.
- ▶ Consider accessibility.

## Bonus: “With great power comes great responsibility” Introducing Boto

- ▶ “I have a workshop coming up where I want 30 people to have virtual machines with custom DNS names. Setting up each one takes a couple of minutes and I am very, very lazy.”
- ▶ See `spawn.py`