maceoutliner Documentation

Release 0.1

Daniel Andrlik

Jun 03, 2018

Contents

Contents:

CHAPTER 1

Install

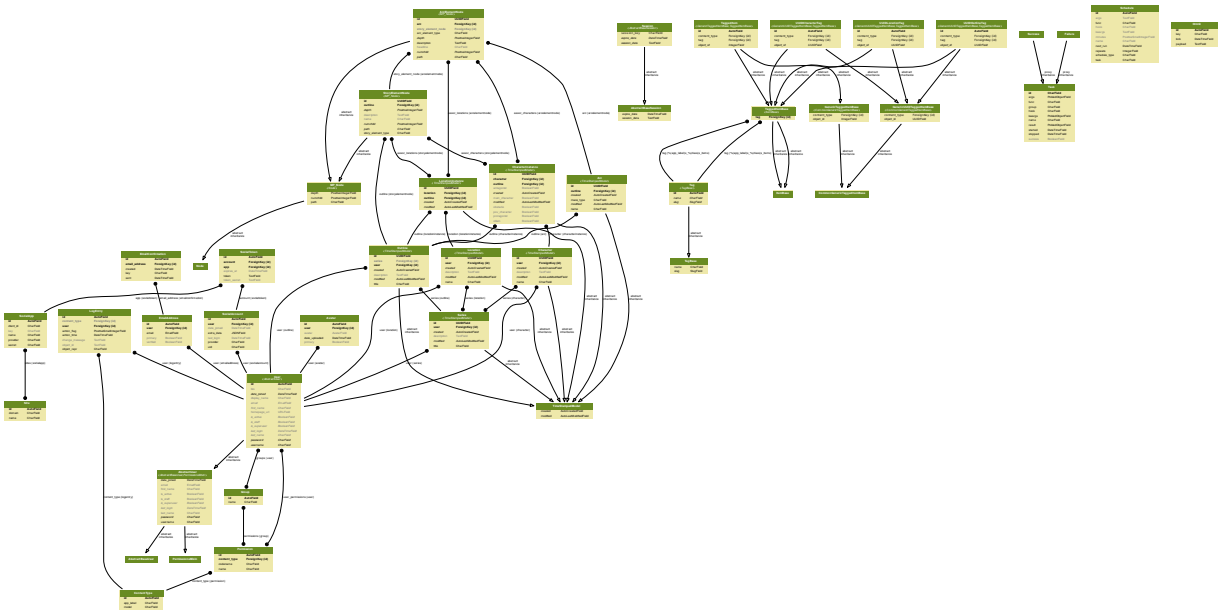
This is where you write how to get a new laptop to run this project.

CHAPTER 2

Model Relationships in MACEOutliner

The MACEOutliner project is made up a number of Django apps, each with a specific focus. What this project provides is a unified user interface, API, and utilities for managing all the disparate pieces of functionality.

A diagram of the model relationships can be found in the figure below.



CHAPTER 3

Deploy

This is where you describe how the project is deployed in production.

CHAPTER 4

Developing with Docker

You can develop your application in a Docker container for simpler deployment onto bare Linux machines later. This instruction assumes an Amazon Web Services EC2 instance, but it should work on any machine with Docker > 1.3 and Docker compose installed.

4.1 Setting up

Docker encourages running one container for each process. This might mean one container for your web server, one for Django application and a third for your database. Once you're happy composing containers in this way you can easily add more, such as a Redis cache.

The Docker compose tool (previously known as fig) makes linking these containers easy. An example set up for your Cookiecutter Django project might look like this:

```
webapp/ # Your cookiecutter project would be in here
    Dockerfile
    ...
database/
    Dockerfile
    ...
webserver/
    Dockerfile
    ...
production.yml
```

Each component of your application would get its own Dockerfile. The rest of this example assumes you are using the base postgres image for your database. Your database settings in *config/base.py* might then look something like:

```
DATABASES = {
        'default': {
            'ENGINE': 'django.db.backends.postgresql_psycopg2',
            'NAME': 'postgres',
            'USER': 'postgres',
```

(continues on next page)

(continued from previous page)

```
            'HOST': 'database',
            'PORT': 5432,
        }
    }
```

The Docker compose documentation explains in detail what you can accomplish in the *production.yml* file, but an example configuration might look like this:

```
database:
    build: database
webapp:
    build: webapp:
    command: /usr/bin/python3.4 manage.py runserver 0.0.0.0:8000 # dev setting
    # command: gunicorn -b 0.0.0.0:8000 wsgi:application # production setting
    volumes:
        - webapp/your_project_name:/path/to/container/workdir/
    links:
        - database
webserver:
    build: webserver
    ports:
        - "80:80"
        - "443:443"
    links:
        - webapp
```

We'll ignore the webserver for now (you'll want to comment that part out while we do). A working Dockerfile to run your cookiecutter application might look like this:

```
FROM ubuntu:14.04
ENV REFRESHED_AT 2015-01-13

# update packages and prepare to build software
RUN ["apt-get", "update"]
RUN ["apt-get", "-y", "install", "build-essential", "vim", "git", "curl"]
RUN ["locale-gen", "en_GB.UTF-8"]

# install latest python
RUN ["apt-get", "-y", "build-dep", "python3-dev", "python3-imaging"]
RUN ["apt-get", "-y", "install", "python3-dev", "python3-imaging", "python3-pip"]

# prepare postgreSQL support
RUN ["apt-get", "-y", "build-dep", "python3-psycopg2"]

# move into our working directory
# ADD must be after chown see http://stackoverflow.com/a/26145444/1281947
RUN ["groupadd", "python"]
RUN ["useradd", "python", "-s", "/bin/bash", "-m", "-g", "python", "-G", "python"]
ENV HOME /home/python
WORKDIR /home/python
RUN ["chown", "-R", "python:python", "/home/python"]
ADD ./ /home/python

# manage requirements
ENV REQUIREMENTS_REFRESHED_AT 2015-02-25
RUN ["pip3", "install", "-r", "requirements.txt"]
```

(continues on next page)

(continued from previous page)

```
# uncomment the line below to use container as a non-root user
USER python:python
```

Running *sudo docker-compose -f production.yml build* will follow the instructions in your *production.yml* file and build the database container, then your webapp, before mounting your cookiecutter project files as a volume in the webapp container and linking to the database. Our example yaml file runs in development mode but changing it to production mode is as simple as commenting out the line using *runserver* and uncommenting the line using *gunicorn*.

Both are set to run on port *0.0.0.0:8000*, which is where the Docker daemon will discover it. You can now run *sudo docker-compose -f production.yml up* and browse to *localhost:8000* to see your application running.

4.2 Deployment

You'll need a webserver container for deployment. An example setup for Nginx might look like this:

```
FROM ubuntu:14.04
ENV REFRESHED_AT 2015-02-11

# get the nginx package and set it up
RUN ["apt-get", "update"]
RUN ["apt-get", "-y", "install", "nginx"]

# forward request and error logs to docker log collector
RUN ln -sf /dev/stdout /var/log/nginx/access.log
RUN ln -sf /dev/stderr /var/log/nginx/error.log
VOLUME ["/var/cache/nginx"]
EXPOSE 80 443

# load nginx conf
ADD ./site.conf /etc/nginx/sites-available/your_cookiecutter_project
RUN ["ln", "-s", "/etc/nginx/sites-available/your_cookiecutter_project", "/etc/nginx/
↪sites-enabled/your_cookiecutter_project"]
RUN ["rm", "-rf", "/etc/nginx/sites-available/default"]

#start the server
CMD ["nginx", "-g", "daemon off;"]
```

That Dockerfile assumes you have an Nginx conf file named *site.conf* in the same directory as the webserver Dockerfile. A very basic example, which forwards traffic onto the development server or gunicorn for processing, would look like this:

```
# see http://serverfault.com/questions/577370/how-can-i-use-environment-variables-in-
↪nginx-conf#comment730384_577370
upstream localhost {
    server webapp_1:8000;
}
server {
    location / {
        proxy_pass http://localhost;
    }
}
```

Running *sudo docker-compose -f production.yml build webserver* will build your server container. Running *sudo docker-compose -f production.yml up* will now expose your application directly on *localhost* (no need to specify the

port number).

4.3 Building and running your app on EC2

All you now need to do to run your app in production is:

- Create an empty EC2 Linux instance (any Linux machine should do).
- Install your preferred source control solution, Docker and Docker compose on the news instance.
- Pull in your code from source control. The root directory should be the one with your *production.yml* file in it.
- Run *sudo docker-compose -f production.yml build* and *sudo docker-compose -f production.yml up*.
- Assign an Elastic IP address to your new machine.
- Point your domain name to the elastic IP.

Be careful with Elastic IPs because, on the AWS free tier, if you assign one and then stop the machine you will incur charges while the machine is down (presumably because you're preventing them allocating the IP to someone else).

4.4 Security advisory

The setup described in this instruction will get you up-and-running but it hasn't been audited for security. If you are running your own setup like this it is always advisable to, at a minimum, examine your application with a tool like OWASP ZAP to see what security holes you might be leaving open.

CHAPTER 5

Indices and tables

- genindex
- modindex
- search