SpisTresci Documentation

Release 0.1

Krzysztof Szumny

July 06, 2016

Contents

Contents:

CHAPTER 1

Install

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

CHAPTER 2

Deploy

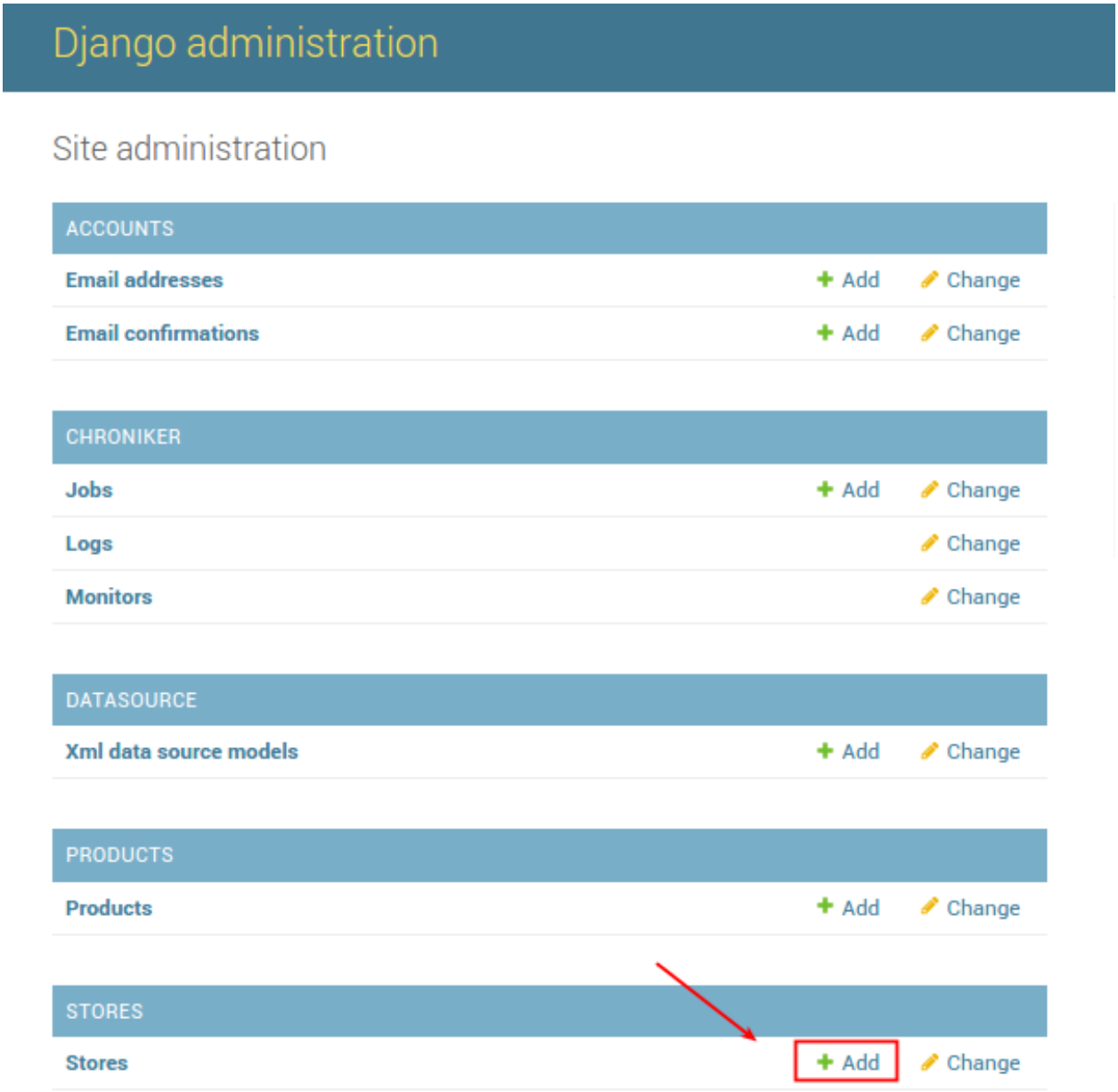
This is where you describe how the project is deployed in production.

CHAPTER 3

Configuration

3.1 Adding new Store

To create new *Store*, you have to have account with admin priviliges. Then, log in to admin pannel (<your website>/admin/) and go to *Stores* section.



Properties:

Enabled If checked, Store will be updated according to schedule of defined jobs

Name Name of store

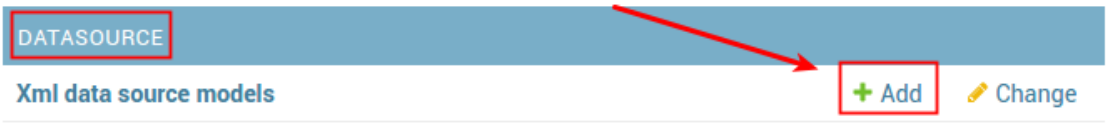
Url Url to main site of store. This is not an url to XML or API. This should be address to main site of *Store*.

DataSource DataSource is responsible for providing information how to import data for specific *Store*. To have a possibility to choose any DataSource from dropdown list, you have to first define new DataSource in Admin Panel, as it is described below. Probably you will have to create seperate DataSource for each Store.

3.2 Adding new DataSource

Right now matadata about products/offers can be imported from XML files. However architecture of SpisTresci supports multiple formats of input data. If you need a support for different format of API, please create an Issue on our github. You can also provide a support for new formats on your own by providing custom classes which will be derived from DataSourceModel and DataSourceImpl classes.

To create DataSource for new Store, go to admin panel and click "+Add" next to DataSource of desired type.



Properties:

Name Each type of DataSource should have own unique name. All other properties are specific for different types of DataSources. Please read documentation for specific DataSource type for more details.

3.3 XMLDataSource

Data Source type XMLDataSource is capable of extracting data not only from single XML file, but also from archives which contains multiple XML files. With *Data Source type* you can specify behaviour for file downloaded from specified *url*.

Single XML The most basic case, when *Store* expose all offers by single XML file as API

Url Address from which data will fetched periodically

Offers root xpath XPath to element which children are offers elements.

For document below, that would be `/root/book`

```
<root>
    <book></book>
    <book></book>
</root>
```

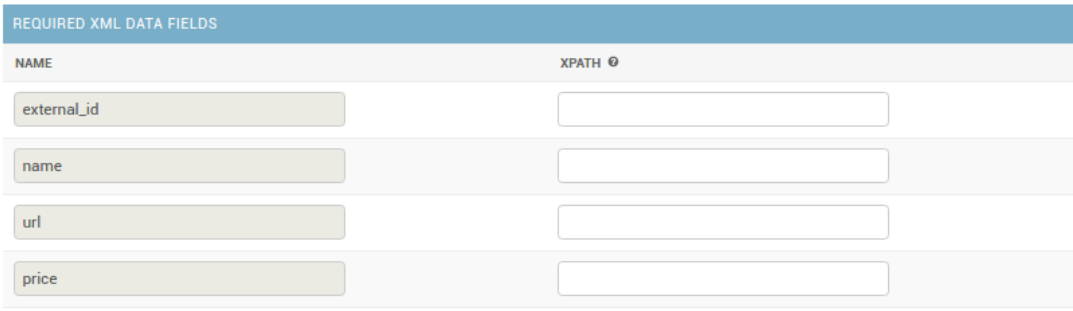
and in that case, that would be `/root/offers/offer`

```
<root>
    <store>
        <location></location>
    </store>
    <offers>
        <offer></offer>
        <offer></offer>
    </offers>
</root>
```

Custom Class Sometimes data provided by Store do not suit very well to assumptions which need to be made during design of database. For example, we assumed that each offer has unique integer *id* in *Store* database, or each offer has name no longer than 256 characters. For sure there are Stores, which can have offers with even longer names, or Stores which have alphanumeric ids.

In such cases, there is no other choice than write some additional code, which will handle those specific cases in desired way. You can find examples of such classes written in Python in spistresci/datasource/specific directory. Your new custom class should derived from *DataSourceImpl* (directly or indirectly).

3.4 XMLDataFields



Required XML Data Fields

Right now there are exactly four required *XML Data Fields* - *external_id*, *name*, *price*, *url*. That means that you have to provide information (by xpath), how to extract those offer metadata. If *Store* which you want to add do not have any of *Required XML Data Fields*, there is no other way - you have to write your own *Custom Class* to hadle such weird case.

external_id is an *id* of offer which *Store* uses in own database to identify specific offer (name of offer is not the best candidate for being a unique identifier, because there can be multiple offers with the same name).

name (default='') Because offers have to be presented somehow to users, that is why we need something like *name* for each offer.

If xpath will not be properly resolved, default value will be used.

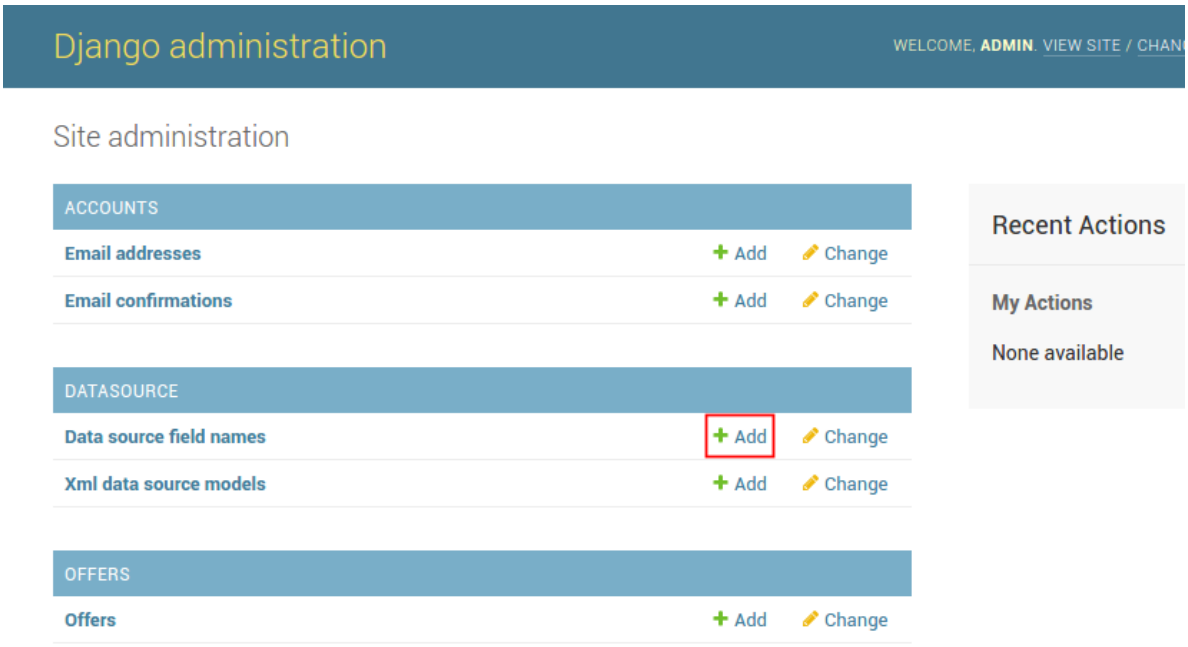
price (default=Decimal('0.00')) Each offer should have own price. If xpath will not be properly resolved, default value will be used.

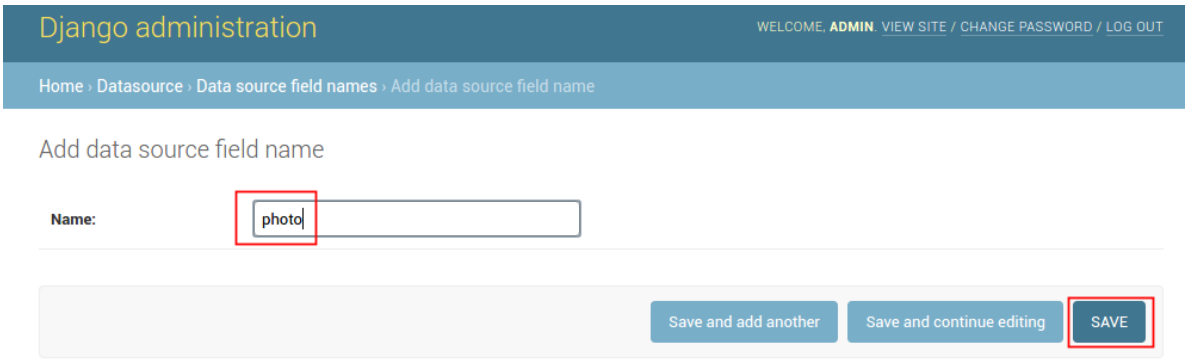
If xpath will not be properly resolved, default value will be used.

url (default='') We assume, that each offer has own url, where you can find details about it.

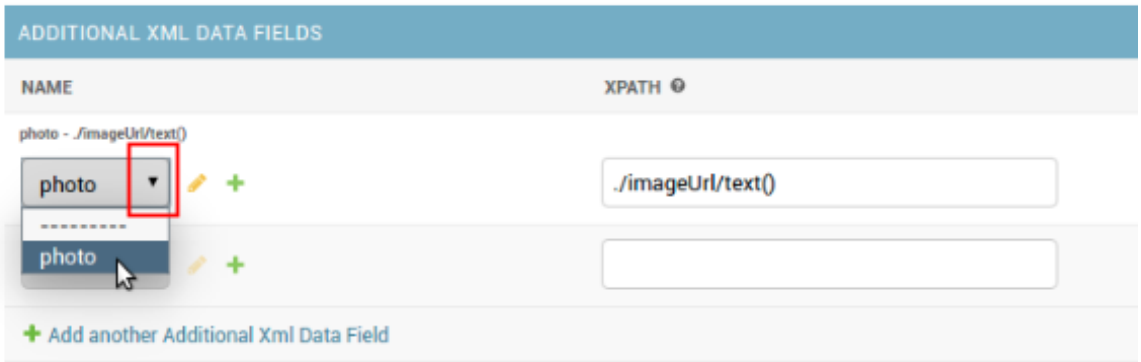
If xpath will not be properly resolved, default value will be used.

Additional XML Data Fields The great news is that you can store any data about offers/products in the database! :) The only thing which you have to do to choose custom datafield name from dropdown is **create new** *Data source field name*.





and then you will be able to choose this name from dropdown list and provide an information how to extract value of this property from XML document (by *xpath*)



For example, to store information about *size* of product in your database, just create new field with name *size* (or 'dimensions' if you prefer - name of property do not have to be exactly the same as it is in XML document of specific store). You will be able to fetch all additional data stored in database via API.

3.5 XMLDataFields - XPath

XPath (XML Path Language) is a best way to specify how to exctract data from XML document. Let's take a look on few examples. Having fallowing XML Document:

```
<document>
  <company>
    <ceo>Elon Musk</ceo>
    <employees>13058</employees>
    <address>
      <city>Palo Alto</city>
      <state>California</state>
      <country>USA</country>
    </address>
  </company>
  <offers>
    <offer avail="0">
      <id>1</id>
      <model>Tesla Roadster</model>
      <imageUrl>https://www.teslamotors.com/sites/default/files/styles/blog-picture_2x_1400xvar_/publ
    </offer>
    <offer avail="1">
      <id>2</id>
      <model>Tesla Model S</model>
      <price>63400.00</price>
      <offerUrl>https://www.teslamotors.com/models</offerUrl>
      <imageUrl>https://www.teslamotors.com/tesla_theme/assets/img/models/section-initial.jpg</imageU
    </offer>
    <offer avail="1">
      <id>3</id>
      <model>Tesla Model X</model>
      <price>69300.00</price>
      <offerUrl>https://www.teslamotors.com/modelx</offerUrl>
      <imageUrl>https://www.teslamotors.com/tesla_theme/assets/img/modelx/section-exterior-profile.jp
    </offer>
    <offer avail="1">
      <id>4</id>
      <model>Tesla Model 3</model>
      <price>35000.00</price>
      <offerUrl>https://www.teslamotors.com/model3</offerUrl>
      <imageUrl>https://www.teslamotors.com/sites/default/files/images/model-3/gallery/gallery-1.jpg<
    </offer>
  </offers>
</document>
```

with xpath `/document/offers/offer/model/text()` you will get `['Tesla Roadster', 'Tesla Model S', 'Tesla Model X', 'Tesla Model 3']`, and similarly with `/document/offers/offer/price/text()` you will get `['63400.00', '69300.00', '35000.00']` (please notice that we got only 3 prices, because 'Tesla Roadster' is not available and document do not describe it's price).

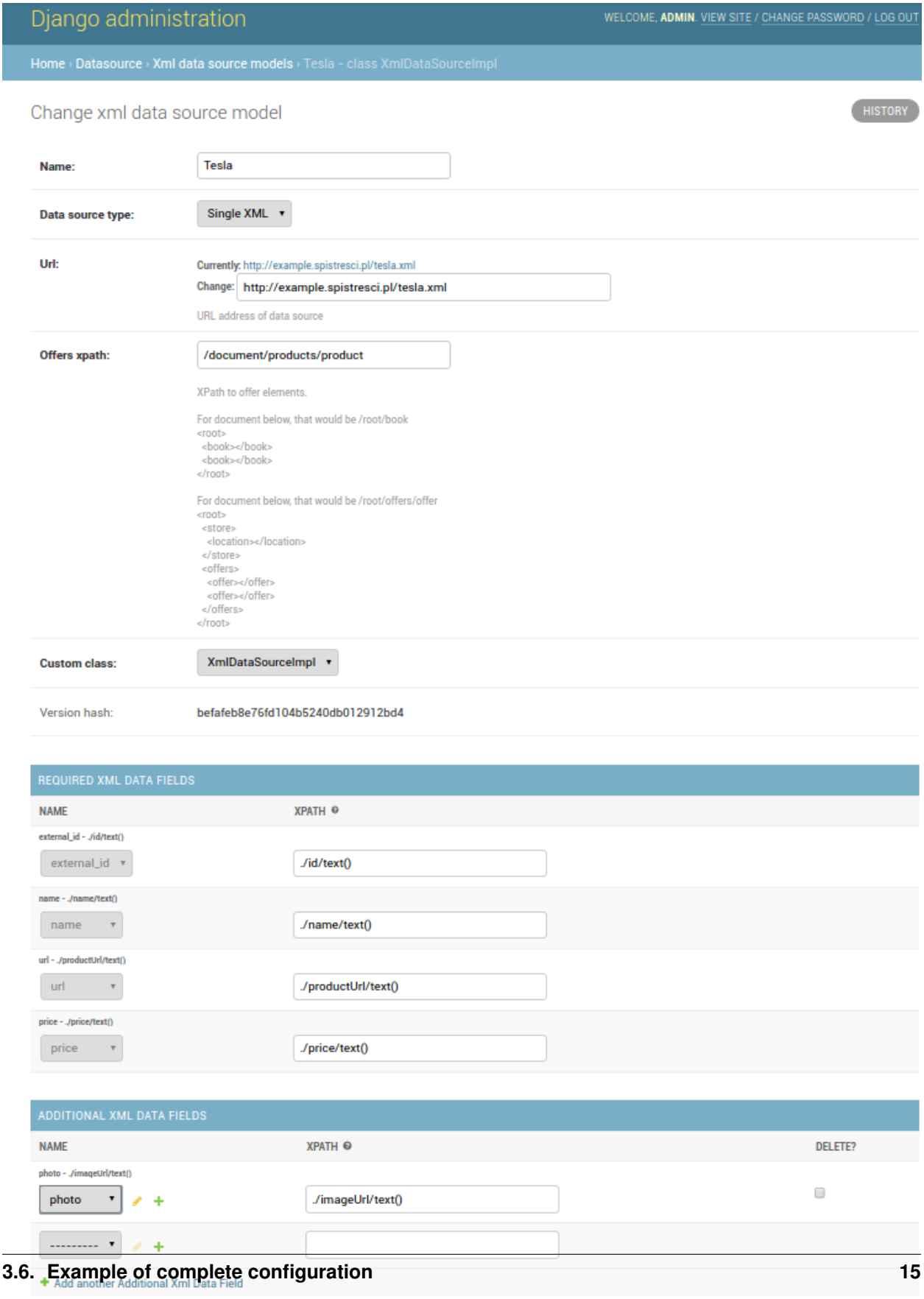
Because of the structure of typical XML document like this, part `/document/offers/offer` may seem to be redundant. Actually, it is very important, because without it alghorithm would not know how to group extracted properties into properties of single offer.

To overcome this problem in that case `/document/offers/offer` should be specified as *offers root xpath* for whole XMLDataSource.

Thanks to that, all *XML Data Field*'s xpaths can be simplified and replaced **with relative xpaths**. In that case that

would be: `./model/text()`, `./price/text()`.

3.6 Example of complete configuration



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
    ...
docker-compose.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/common.py* might then look something like:

```
DATABASES = {
        'default': {
            'ENGINE': 'django.db.backends.postgresql_psycopg2',
            'NAME': 'postgres',
            'USER': 'postgres',
            'HOST': 'database',
            'PORT': 5432,
        }
    }
```

The Docker compose documentation explains in detail what you can accomplish in the *docker-compose.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"]

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

Running *sudo docker-compose build* will follow the instructions in your *docker-compose.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 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/yo
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#comme
upstream localhost {
    server webapp_1:8000;
}
server {
    location / {
        proxy_pass http://localhost;
    }
}
```

Running *sudo docker-compose build webserver* will build your server container. Running *sudo docker-compose 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 *docker-compose.yml* file in it.
- Run *sudo docker-compose build* and *sudo docker-compose 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