Booktype Documentation

Release 2.4.0

Aleksandar Erkalović

May 08, 2018

Contents

1 Deployment **3**
- 1.1 Booktype Project structure . . . 3
- 1.2 Profiles . . . 4
- 1.3 Booktype settings . . . 5
- 1.4 Redis settings . . . 8
- 1.5 PostgreSQL . . . 9
- 1.6 Createbooktype . . . 10
- 1.7 Upgrade instructions . . . 12
- 1.8 MOBI . . . 15
- 1.9 Installing Booktype on CentOS . . . 15
- 1.10 Installing Booktype on Mac OS X . . . 19
- 1.11 Running Booktype on the Nginx web server . . . 21
- 1.12 As Gunicorn . . . 22
- 1.13 As FastCGI . . . 22

2 Development **25**
- 2.1 How to contribute . . . 25
- 2.2 Before you send a pull request . . . 26
- 2.3 Python Style . . . 27
- 2.4 JavaScript Style . . . 29
- 2.5 Development profile . . . 31
- 2.6 Booktype core testing . . . 32
- 2.7 Documenting . . . 35

3 Deployment **39**

4 Development **41**

5 Community **43**

6 Indices and tables **45**

Welcome! This is the documentation for contributors to the Booktype project.

CHAPTER 1

Deployment

1.1 Booktype Project structure

1.1.1 What is a Project

In simple words, each Booktype project is a separate web site. Most of the users would like to customise their site, develop their own front page, design their own look or install custom plugins. Because of that each installation of Booktype is a new project which allows easy customisation and simple upgrade to the latest version of Booktype.

To achieve that goal we have something called *Booktype project*. Configuration files and scripts are autogenerated for each project specifically. User has to define minimal set of configuration options and everything else is ready to run out of the box.

1.1.2 File structure

This is structure on disk for project called *bkdev*:

```
bkdev/
    booktype.env
    manage.py

    conf/
        wsgi.apache
        gunicorn.nginx
        fastcgi.nginx

    lib/

    static/
    data/

    bkdev_site/
```

(continues on next page)

(continued from previous page)

```
        locale/
        templates/
        static/
        wsgi.py

        settings/
            base.py
            dev.py
            prod.py

        urls/
            dev.py
            prod.py
```

- **booktype.env** - It is used for
- **manage.py** - This is our manage file
- **conf/** - Autogenerated configuration files for external services.
  - **wsgi.apache** - Apache site definition.
  - **gunicorn.nginx** - Gunicorn.
  - **fastcgi.nginx** - FastCGI.
- **lib/** - Location for non web related code or 3rd party libraries.
- **static/** - Location for auto collected static files.
- **data/** - Location for user uploaded content.
- **bkdev_site/** - Autogenerated Django application for out project.
  - **wsgi.py** - Autogenerated wsgi file for our project.
  - **settings/** - Autogenerated base settings.
    - **base.py** - Autogenerated base settings.
    - **dev.py** - Autogenerated settings for development profile.
    - **prod.py** - Autogenerated settings for production profile.
  - **urls** - URL dispatcher for our project.
    - **dev.py** - For development profle.
    - **prod.py** - For production profile.

1.2 Profiles

1.2.1 What are they and why we need them

Project could be deployed using different profiles. Why would we need them in the first place?

Let us imagine we were hired by a community of writters to extend their Booktype installation with additional features.

Our first step would be to create *Booktype project*. This project would be used as a base for our future development. When we are working and developing new features on our local machine we would use **development profile**. When we are happy with new changes we would like to show them and test them somewhere. This is where our **staging**

profile comes. That is for instance profile we use on a staging machine where our customers can test new features. When they are happy with what we have done we deploy the same project to production using **production profile**.

This means our project can be deployed using different settings depending if we are developing it on our local machine or running it in the production. In our **development profile** we can include additional django debug applications, including Django admin application, disable access to some public API our custom code is using or use different backend for storing data. In our **production profile** system we can disable all the debugging options, extra logging we needed during the development phase and allow access to public API our custom code needs.

As you can see, profiles are just different Django settings files in which you configure your application for different environments.

1.2.2 Custom profile

Each profile is a new Django settings file.

We want to create new profile "maintanance". There are better ways how to do it but in our over simplified example we will create new profile which will define new set of configuration options and files to show one static page "Please wait. Site in maintanance mode.".

First we need new settings file in *$PROJECT/settings/maintanance.py*. Make it by copying content of already existing profile dev.

Then change this options in the settings:

```
ROOT_URLCONF = '{}.urls.maintanance'.format(BOOKTYPE_SITE_DIR)
```

Then we need to change existing *manage.py* file in the project root to use new profile.

```
os.environ.setdefault("DJANGO_SETTINGS_MODULE", "$PROJECT.settings.maintanance")
```

At the end we need to define our URL dispatcher in *$PROJECT/url/maintanance.py* file.

```
from django.conf.urls import patterns, url, include
from django.views.generic import TemplateView

urlpatterns = patterns('',
    (r'^$', TemplateView.as_view(template_name="our_message.html")),
)
```

1.3 Booktype settings

- *Booktype specific*
- *Redis*
- *Rest of the settings*
- *Deprecated*

Warning: Work in progress.

1.3.1 Booktype specific

BOOKTYPE_SITE_NAME

You can name your Booktype instance. This name will be used in default templates as the name of your site.

Example:

```
BOOKTYPE_SITE_NAME = 'My Booktype instance'
```

BOOKTYPE_SITE_DIR

This is the directory name of your Booktype project on disk. It always ends with "_site".

Example:

```
BOOKTYPE_SITE_DIR = 'test_site'
```

Note: Createbooktype script will configure this value appropriately. User will need to change this value only when changing or renaming Booktype project layout on disk.

BOOKTYPE_ROOT

Full path pointing to location of Booktype project.

Example:

```
BOOKTYPE_ROOT = '/var/www/mybooktype'
```

Note: Createbooktype script will configure this value appropriately. User will need to change this value only when changing location of Booktype project.

BOOKTYPE_URL

URL pointing to your Booktype project.

Example:

```
BOOKTYPE_URL = 'http://booktype.mydomain.com'
```

Note: Other URL settings will be configured using this value as a base URL.

PROFILE_ACTIVE

Each Booktype instance has more then one profile. Profiles are used so we can have different settings and environment for our Booktype instance in production mode, development mode or staging mode. Different profiles have different settings file and this value is configured accordingly in setting file.

Example for development profile:

```
PROFILE_ACTIVE = 'dev'
```

Note: There is no need for user to change this settings unless new profile is being created.

DATA_ROOT

Full path where user uploaded data is. Default value is using *BOOKTYPE_ROOT* to generate this settings.

Note: Change this only if user data is not at the same location as Booktype project.

DATA_URL

Full URL where user uploaded data is. Default value is using *BOOKTYPE_URL* to generate this settings.

Note: Change this only if user data is not at the same URL as Booktype project.

COVER_IMAGE_UPLOAD_DIR

Default: `"cover_images/"`.

Name of the directory where cover images are placed. Base directory is *DATA_ROOT*.

PROFILE_IMAGE_UPLOAD_DIR

Default: `"profile_images/"`

Name of the directory where profile images are stored. Base directory is *DATA_ROOT*.

BOOKTYPE_CONVERTER_MODULES

List of plugins for book conversions into different formats.

Example:

```
BOOKTYPE_CONVERTER_MODULES = (
    'booktype.convert.converters',
)
```

1.3.2 Redis

Configuration for connecting to Redis database.

REDIS_HOST

Default: `"localhost"`.

REDIS_PORT

Default: `6379`.

REDIS_DB

Default: `1`.

REDIS_PASSWORD

Default: `None`.

1.3.3 Rest of the settings

COMPRESS_ENABLED

Used for configuring `django_compressor` application. "Dev" profile has compression disabled by default.

COMPRESS_OFFLINE

Used for configuring `django_compressor` application. "Dev" profile has compression disabled by default.

1.3.4 Deprecated

We don't use these settings anymore but they are still here for compatibility issues.

BOOKI_NAME

BOOKI_ROOT

BOOKI_URL

THIS_BOOKI_SERVER

BOOKI_MAINTENANCE_MODE

1.4 Redis settings

Redis configuration (on Debian based systems) is in /etc/redis/redis.conf. Here is a list of configuration options you might want to change.

Link to offical Redis documentation: http://redis.io/topics/config .

1.4.1 save

Example:

```
save 900 1
save 300 10
save 60 10000
```

If you don't want Redis to save data to disk comment (or remove) all save options. Default values are too low for high traffic Booktype site.

1.4.2 appendfsync

Example:

```
# appendfsync always
appendfsync everysec
# appendfsync no
```

This option works only if you are saving data to disk. Choose "no" or "everysec" if you have a lot of traffic on the site.

1.5 PostgreSQL

1.5.1 How to create PostgreSQL database

Check the documentation: https://docs.djangoproject.com/en/dev/ref/databases/ .

1. Install PostgreSQL and Python modules:

   ```
   $ apt-get install  postgresql python-psycopg2
   ```

2. Change the password for PostgreSQL. This you need to do `ONLY` if you don't already know the password:

   ```
   $ sudo su postgres -c psql template1
   template1=# ALTER USER postgres WITH PASSWORD 'password';
   template1=# \q

   $ sudo passwd -d postgres
   $ sudo su postgres -c passwd
   ```

3. Create PostgreSQL user "booktype":

   ```
   $ sudo su postgres -c createuser booktype
   ```

4. Create database named "booktype" where user "booktype" is owner. This could depend of your OS version and can ask for another template:

   ```
   $ createdb -D template1 -E utf8 -O booktype booktype
   ```

5. Allow connections to database booktype for user booktype. This can depend of your requirements:

```
$ vi /etc/postgresql/*/main/pg_hba.conf  (full file name depends of PostgreSQL␣
↪version)

local   booktype    booktype                        md5
```

6. Restart PostgreSQL:

```
$ service postgresql restart
```

1.5.2 How to do it on Ubuntu

The following instructions were tested on Ubuntu Lucid 10.04 and slightly differ from the generic instructions above.

1. Install PostgreSQL and Python modules:

```
$ sudo apt-get install postgresql python-psycopg2
```

2. Change the password for PostgreSQL. This you need to do `ONLY` if you don't already know the password:

```
$ sudo -u postgres psql postgres
postgres=# \password postgres
postgres=# \q

$ sudo passwd -d postgres
$ sudo su postgres -c passwd
```

3. Create PostgreSQL user "booktype":

```
$ sudo -u postgres createuser -SDR booktype
```

4. Create database named "booktype" where user "booktype" is owner:

```
$ sudo -u postgres createdb -E utf8 -O booktype booktype
```

5. Allow connections to database booktype for user booktype:

```
$ sudo nano /etc/postgresql/8.4/main/pg_hba.conf  (exact file name depends on␣
↪PostgreSQL version)

local   booktype    booktype                          md5
```

6. Restart PostgreSQL:

```
$ sudo invoke-rc.d postgresql-8.4 restart
```

1.6 Createbooktype

Createbooktype is Booktype's command-line utility for creating *Booktype projects* on the disk with auto generated configuration files.

1.6.1 Usage

```
createbooktype [--database <type>|--profile <type>|--check-versions] <project
↪location>
```

<project location> can be full path to project location or name of the project.

Example:

argument	project name
/var/www/mybk	mybk
bk20	bk20
../bk20	bk20

Note: User must have write permissions to create or write to the project location. If project location already exists it will just create the needed files.

1.6.2 Options

```
--help
```

Show help message and exit.

```
--quiet
```

Do not show any messages.

```
--verbose
```

Show messages.

```
--database <type>
```

Configure *Project* to use specific database backend. Options are: “postgres”, “postgresql”, “sqlite”.

```
--profile <type>
```

Configure *Project* to use profile <type>. Options are: “dev”, “prod”.

```
--check-versions
```

Check versions of packages

```
--virtual-env VIRTUAL_ENV
```

Specifies the default VIRTUAL_ENV

1.6.3 Examples of usage

This will create project called **bk2** in current directory. It will use PostgreSQL database as a backend and automatically be set i$

```
$ ./scripts/createbooktype bk2
```

This will create project called **mybk** in /var/www/ directory. It will use Sqlite3 as a backend and will be set in development prof$

```
$ ./scripts/createbooktype --database sqlite3  --profile dev /var/www/mybk/
```

1.7 Upgrade instructions

1.7.1 Booktype 1.6.1

Templates have been changed a bit in this release. Check lib/booki/editor/templates/editor/ directory. Structure has been changed for:

- edit_book.html
- edit_header.html
- edit_sidebar.html
- edit_content.html
- edit_templates.html
- tab_chapters.html
- tab_history.html
- tab_notes.html
- tab_publish.html
- tab_settings.html
- tab_version.html

1.7.2 Booktype 1.6.0

There are changes in the database schema and database migration is required.

1.7.3 Booktype 1.5.5

There are changes in the database schema and database migration is required.

Update your project settings.py file. If you are using default settings for Objavi you should update them:

```
OBJAVI_URL = "http://objavi.booktype.pro/"
ESPRI_URL = "http://objavi.booktype.pro/espri"
```

1.7.4 Booktype 1.5.4

There are changes in the database schema and database migration is required.

1.7.5 Booktype 1.5.3

Update your project settings.py file to use messaging framework. You have to:

```
- Add new options

   MESSAGE_STORAGE = 'django.contrib.messages.storage.session.SessionStorage'
   TEMPLATE_CONTEXT_PROCESSORS = ('django.contrib.auth.context_processors.auth',
                                  'django.contrib.messages.context_processors.messages
↪')

- Add new messaging middleware to the list:
   MIDDLEWARE_CLASSES = (...,
                         'django.contrib.messages.middleware.MessageMiddleware',
                         ...)

- Add new Django app to the list:
   INSTALLED_APPS = (...,
                     'django.contrib.messages',
                     ...)
```

Notice: All of these changes will require "django-admin migrate" at the end.

Upgrade your config files to include Control Center:

```
- Upgrade settings.py file with:
  INSTALLED_APPS = (...,
                    'booktypecontrol'
                    ,...)


- Add to the end of settings.py file

  from booki.utils import config

  try:
    BOOKTYPE_CONFIG = config.loadConfiguration()
  except config.ConfigurationError:
    BOOKTYPE_CONFIG = {}

- Template file lib/booki/portal/templates/base.html has been modified.
```

Notice: All of these changes will require "django-admin migrate" at the end.

Style of database configuration has been changed so please update your configuration. This is a normal Django database configuration and please check Django documentation for more information and options.

It used to be:

```
DATABASE_ENGINE = 'postgresql_psycopg2'
DATABASE_NAME = ''
DATABASE_USER = ''
DATABASE_PASSWORD = ''
DATABASE_HOST = 'localhost'
DATABASE_PORT = ''
```

Now it is:

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

New configuration for load Django templates. Please change your configuration:

```
import django

if django.VERSION[1] < 3:
    TEMPLATE_LOADERS = (
                          'django.template.loaders.filesystem.load_template_source',
                          'django.template.loaders.app_directories.load_template_source
↪',
                          'django.template.loaders.eggs.load_template_source',
                        )
else:
    TEMPLATE_LOADERS = (
                          'django.template.loaders.filesystem.Loader',
                          'django.template.loaders.app_directories.Loader',
                          'django.template.loaders.eggs.Loader',
                        )
```

1.7.6 Booktype 1.5.2

Update your project settings.py. You have to add new middleware called "LocaleMiddleware" to the list.:

```
MIDDLEWARE_CLASSES = (
    'django.middleware.common.CommonMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.locale.LocaleMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.middleware.transaction.TransactionMiddleware'
)
```

Update your project settings.py. You don't have to but you can comment LANGUAGES options.:

```
LANGUAGE_CODE = 'en-us'

# Uncomment this if you want to limit language translations only to specific list of␣
↪languages
#
# gettext = lambda s: s
#
# LANGUAGES = (
#   ('en-us', gettext('English'))
# )
```

By default "createbooktype" script will now create "locale" directory in your Booktype project.

Update your project settings.py:

```
LOCALE_PATHS = (
    '%s/locale' % os.path.dirname(booki.__file__),
)
```

1.8 MOBI

Booktype can export books in MOBI format using external 3rd party software. User has choice between Calibre (GNU General Public License v3) and KindleGen from Amazon. Calibre is enabled by default.

User is required to install 3rd party software on its own.

1.8.1 Use Calibre

Configuration options:

```
MOBI_CONVERT = "calibre"
CALIBRE_PATH = "ebook-convert"
CALIBRE_ARGS = ""
```

MOBI_CONVERT instructs Booktype to use Calibre for MOBI export.

CALIBRE_PATH defines full path to the command line tool used for conversion. By default it is configured as if Calibre is installed on the system and globally accessible. In case it is not, user can define full path to the ebook-convert tool.

CALIBRE_ARGS defines extra arguments for the ebook-convert tool. Extra arguments are available here: http://manual.calibre-ebook.com/cli/ebook-convert.html.

1.8.2 Use KindleGen

Configuration options:

```
MOBI_CONVERT = "kindlegen"
KINDLEGEN_PATH = "kindlegen"
```

MOBI_CONVERT instructs Booktype to use KindleGen for MOBI export.

KINDLEGEN_PATH defines full path to the command line tool used for conversion. By default it is configured as if KindleGen is installed on the system and globally accessible. In case it is not, user can define full path to the kindlegen tool.

1.9 Installing Booktype on CentOS

Instructions were tested on CentOS 6.3.

Be careful you have correct access permissions. We assume your Python Virtual Environment is called 'mybooktype' and your Booktype project is called 'mybook'. Feel free to change it.

Before you start installing check your documentation how to add EPEL repository! For instance:

```
su -c 'rpm -Uvh http://download.fedoraproject.org/pub/epel/6/i386/epel-release-6-7.
↪noarch.rpm'
```

You will also need to install "Development Tools":

```
su -c 'yum groupinstall "Development Tools"'
```

1.9.1 Booktype with Sqlite3

This example works with Sqlite3 and Python built in web server. This method is not recommended for production server.

How to install

```
# Install needed packages
su -c 'yum -y install python python-devel sqlite git python-virtualenv python-pip'
su -c 'yum -y install redis libxml2-devel libxslt-devel libjpeg libjpeg-devel zlib␣
↪zlib-devel'

# Create Python Virtual Environment and install needed Python modules
virtualenv --distribute mybooktype
cd mybooktype
source bin/activate
pip install Django==1.3 South==0.7.5 unidecode lxml PIL

# Fetch Booktype source
git clone https://github.com/booktype/Booktype.git

# Create Booktype project
./Booktype/scripts/createbooktype --database sqlite mybook

# Initialise Booktype
source mybook/booktype.env
django-admin.py syncdb --noinput
django-admin.py migrate
django-admin.py loaddata documentation_licenses
django-admin.py createsuperuser
```

Deploy using built in web server (not recommended but good for testing)

```
# This has to be done every time you want to start a server
cd mybooktype
source bin/activate
source mybook/booktype.env
django-admin.py runserver 0.0.0.0:8080
```

Deploy using Apache (recommended)

```
# Install needed packages
yum install httpd mod_wsgi
```

Copy configuration file:

```
cp mybooktype/mybook/wsgi.apache file into /etc/httpd/conf.d/booktype.conf

# User www-data should be the owner
chown -R apache:apache mybooktype/

# Edit configuration file
# You need to add name of the server and change log path to /var/log/httpd/
vi /etc/httpd/conf.d/booktype.conf

service httpd restart
```

If you will get permission errors you might need to disable SELinux support. Please check CentOS documentation how to do that. You can also check any documentation related to CentOS+Apache2+mod_wsgi+Django deployment.

1.9.2 Booktype with PostgreSQL

This example works with PostgreSQL and Python built in web server. Version of PostgreSQL server depends of your distribution. This method is recommended for production server.

How to install

```
# Install needed packages
su -c 'yum -y install python python-devel sqlite git python-virtualenv python-pip'
su -c 'yum -y install redis libxml2-devel libxslt-devel libjpeg libjpeg-devel zlib␣
↪zlib-devel'
su -c 'yum -y install postgresql-server postgresql-libs postgresql-devel python-
↪psycopg2'

# Create Python Virtual Environment and install needed python modules
virtualenv --distribute mybooktype
cd mybooktype
source bin/activate
pip install Django==1.3 South==0.7.5 unidecode lxml PIL psycopg2

# Fetch Booktype source
git clone https://github.com/booktype/Booktype.git

# Create Booktype project
./Booktype/scripts/createbooktype --database postgresql mybook

# Initialize PostgreSQL. ONLY if you did not have PostgreSQL installed before
chkconfig postgresql on
service postgresql initdb
service postgresql start

# Become postgres user
su - postgres

# Create PostgreSQL user and enter password
createuser -SDRP booktype

# Create PostgreSQL database
createdb -E utf8 -O booktype booktype
```

(continues on next page)

(continued from previous page)

```
# Stop being Postgres user
exit
```

You will need to enter database info in the settings file. Edit mybooktype/mybook/settings.py file and put this as database info (you will need to enter username password also).

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': 'booktype',
        'USER': 'booktype',
        'PASSWORD': 'ENTER PASSWORD HERE',
        'HOST': '',
        'PORT': ''
    }
}
```

Allow connections to database booktype for user booktype. This can depend of your requirements. Edit /var/lib/pgsql/data/pg_hba.conf file and put this inside.

```
local   booktype    booktype                        md5
```

Restart PostgreSQL server after this.

```
service postgresql restart
```

You can continue now with initialisation.

```
source mybook/booktype.env
django-admin.py syncdb --noinput
django-admin.py migrate
django-admin.py loaddata documentation_licenses
django-admin.py createsuperuser
```

Start redis

```
chkconfig redis on
service redis start
```

Deploy using built in web server (not recommended but good for testing)

```
# This has to be done every time you want to start a server
cd mybooktype
source bin/activate
source mybook/booktype.env
django-admin.py runserver 0.0.0.0:8080
```

Deploy using Apache (recommended)

```
# Install needed packages
yum install httpd mod_wsgi
```

Copy configuration file:

```
cp mybooktype/mybook/wsgi.apache file into /etc/httpd/conf.d/booktype.conf

# User www-data should be the owner
chown -R apache:apache mybooktype/

# Edit configuration file
# You need to add name of the server and change log path to /var/log/httpd/
vi /etc/httpd/conf.d/booktype.conf

service httpd restart
```

If you will get permission errors you might need to disable SELinux support. Please check CentOS documentation how to do that. You can also check any documentation related to CentOS+Apache2+mod_wsgi+Django deployment.

1.10 Installing Booktype on Mac OS X

Instructions were tested on Mac OS X 10.5, 10.6 and 10.8. We assume your Python Virtual Environment is called 'mybooktype' and your Booktype project is called 'mybook'. Feel free to change it.

Be careful you have correct access permissions.

You **MUST** install Homebrew and you **MUST** install Xcode or Command Line Development Tools.

```
http://mxcl.github.com/homebrew/
```

You **MUST** figure out for yourself how to start Redis and PostgreSQL server installed with Homebrew. Write 'brew info redis' for more information.

1.10.1 Booktype with Sqlite3

This example works with Sqlite3 and Python built in web server. This method is not recommended for production server.

How to install

```
# Install needed packages
brew install git redis libjpeg libpng libxml2

# Create Python Virtual Environment and install needed Python modules
virtualenv --distribute mybooktype
cd mybooktype
source bin/activate

# For Mac OS X 10.8
# There is a problem with 10.8 and you MUST do this. Version of library might be␣
↪different.
pip install lxml --install-option="--with-xml2-config=/usr/local/Cellar/libxml2/2.8.0/
↪bin/xml2-config"

# If you don't have 10.8 you can try to do just this
pip install lxml

# Install rest of Python packages
```

(continues on next page)

(continued from previous page)

```
pip install Django==1.3 South==0.7.5 unidecode PIL

# Fetch Booktype source
git clone https://github.com/booktype/Booktype.git

# Create Booktype project
./Booktype/scripts/createbooktype --database sqlite mybook

# Initialise Booktype
source mybook/booktype.env
django-admin.py syncdb --noinput
django-admin.py migrate
django-admin.py loaddata documentation_licenses
django-admin.py createsuperuser

# Start server
django-admin.py runserver 0.0.0.0:8080
```

How to run it again

```
cd mybooktype
source bin/activate
source mybook/booktype.env
django-admin.py runserver 0.0.0.0:8080
```

1.10.2 Booktype with PostgreSQL

This example works with PostgreSQL and Python built in web server. Version of PostgreSQL server depends of your distribution. This method is recommended for production server.

Be aware this has not been fully tested.

How to install

```
# Install needed packages
brew install git redis libjpeg libpng libxml2 postgresql

# Create Python Virtual Environment and install needed Python modules
virtualenv --distribute mybooktype
cd mybooktype
source bin/activate

# For Mac OS X 10.8
# There is a problem with 10.8 and you MUST do this. Version of library might be␣
↪different.
pip install lxml --install-option="--with-xml2-config=/usr/local/Cellar/libxml2/2.8.0/
↪bin/xml2-config"

# If you don't have 10.8 you can try to do just this
pip install lxml

# Install rest of Python packages
```

(continues on next page)

(continued from previous page)

```
pip install Django==1.3 South==0.7.5 unidecode PIL psycopg2

# Fetch Booktype source
git clone https://github.com/booktype/Booktype.git

# Create Booktype project
./Booktype/scripts/createbooktype --database postgresql mybook

# Create PostgreSQL user and enter password
/usr/local/bin/createuser -SDRP booktype

# Create PostgreSQL database
/usr/local/bin/createdb -E utf8 -O booktype booktype
```

You will need to enter database info in the settings file. Edit mybooktype/mybook/settings.py file and put this as database info (you will need to enter username password also).

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': 'booktype',
        'USER': 'booktype',
        'PASSWORD': 'ENTER PASSWORD HERE',
        'HOST': 'localhost',
        'PORT': ''
    }
}
```

You can continue now with initialisation.

```
source mybook/booktype.env
django-admin.py syncdb --noinput
django-admin.py migrate
django-admin.py loaddata documentation_licenses
django-admin.py createsuperuser

# Run server
django-admin.py runserver 0.0.0.0:8080
```

How to run it again

```
cd mybooktype
source bin/activate
source mybook/booktype.env
django-admin.py runserver 0.0.0.0:8080
```

1.11 Running Booktype on the Nginx web server

Example commands are for a Debian/Ubuntu based system. It is recommended to use Supervisor (http://supervisord.org/) or similar software (Upstart on Ubuntu) to control your Gunicorn or FastCGI processes.

1.12 As Gunicorn

Let us imagine our Booktype project is installed in the */var/www/mybooktype/mybook/* directory. This directory should be owned by the user who is going to run the gunicorn_django process (probably www-data).

```
# Install needed packages
sudo apt-get install nginx

# Copy configuration file
cp /var/www/mybooktype/mybook/gunicorn.nginx /etc/nginx/sites-available/booktype

# Edit configuration file
# You should change: server_name and fastcgi_pass
vi /etc/nginx/sites-available/booktype

# Enable your Booktype site
ln -s /etc/nginx/sites-available/booktype /etc/nginx/sites-enabled/booktype

# Restart Nginx
service nginx restart

# Activate Python Virtual Environment (IF YOU ARE USING IT)
source /var/www/mybooktype/bin/activate

# Install Gunicorn
pip install gunicorn

# Load environment variables
source /var/www/mybooktype/mybook/booktype.env

# Start Gunicorn (Basic example)
gunicorn_django -b 127.0.0.1:8000 -w 4
```

1.13 As FastCGI

Let us imagine our Booktype project is installed in */var/www/mybooktype/mybook/* directory.

```
# User www-data should be owner
sudo chown -R www-data:www-data /var/www/mybooktype/mybook/

# Install needed packages
sudo apt-get install nginx

# Copy configuration file
cp /var/www/mybooktype/mybook/fastcgi.nginx /etc/nginx/sites-available/booktype

# Edit configuration file
# You should change: server_name and fastcgi_pass
vi /etc/nginx/sites-available/booktype

# Enable your Booktype site
ln -s /etc/nginx/sites-available/booktype /etc/nginx/sites-enabled/booktype

# Restart Nginx
```

(continues on next page)

(continued from previous page)

```
service nginx restart

# Activate Python Virtual Environment (IF YOU ARE USING IT)
source /var/www/mybooktype/bin/activate

# Install Flup  (http://www.saddi.com/software/flup/)
pip install flup

# Load environment variables
source /var/www/mybooktype/mybook/booktype.env

# Start FastCGI process (Basic example)
django-admin.py runfcgi host=127.0.0.1 port=8000
```

CHAPTER 2

Development

2.1 How to contribute

2.1.1 How can you help

It is not just about core development. We also need help with translating interface, building new book themes, helping other people on the forum or just help us write the documentation.

Our official tracker is a good place to find already available tasks to work on - http://dev.sourcefabric.org/browse/BK.

2.1.2 Get the source

1. Fork sourcefabric/Booktype repository
    - https://help.github.com/articles/fork-a-repo
2. Clone your fork
3. Create new local feature branch
    - http://learn.github.com/p/branching.html
4. Create pull request with your feature/bugfix
    - https://help.github.com/articles/creating-a-pull-request

2.1.3 What you should know

This documentation is work in progress but it is good place where you can find out more about the project, structure, tools we use and how we organise our work.

2.2 Before you send a pull request

2.2.1 First step

Besides doing the checks manually it is also possible to automatically validate some of the things before sending pull request.

At the moment script does:

- Checks if modified Python code is according to PEP8 specification
- Checks if modified JavaScript code is according to Booktype JavaScript specification
- Creates and initialises Booktype instance
- Executes Booktype tests

```
$ scripts/before_pull_request.sh
```

Usage

```
$ scripts/before_pull_request.sh [-s|-n|-m]
```

-s

Validates staged files in your repository. Result of **git diff –name-only –staged**.

-n

Validates non staged files in your repository. Result of **git diff –name-only**.

-m

Validates modified files. Result of **git status –porcelain**.

Examples

No problems with modified files:

```
BEFORE PULL REQUEST 3000
------------------------------------------------------------
[*] Checking only staged files
    Seems like all the files are according to PEP8 and Booktype JavaScript standard.

Press [Enter] key to start deployment...
```

Some of the files are not according to the specification:

```
BEFORE PULL REQUEST 3000
------------------------------------------------------------
[*] Checking all modified files
    [jshint] /Booktype/lib/booktype/apps/edit/static/edit/js/booktype/covers.js
    [PEP8]   /Booktype/lib/booktype/apps/edit/views.py
    [PEP8]   /Booktype/fabfile.py

    Some of the files are not according to PEP8 or Booktype JavaScript standard. Fix␣
↪the style only in your code.
    Create new ticket if style is broken in someone else's code and fix it later.

    Please check:
        - http://legacy.python.org/dev/peps/pep-0008/
        - Booktype docs 'docs/development/style.rst'
        - Booktype docs 'docs/development/js_style.rst'

Press [Enter] key to start deployment...
```

2.3 Python Style

2.3.1 Style

PEP 8 is the de-facto code style guide for Python. Django also has Coding style document which covers their coding style.

Just like in Django Coding style we don't use 79 characters limit for a line.

2.3.2 How to check

Conforming your Python code to PEP 8 is generally a good idea and helps make code more consistent when working on projects with other developers.

If you don't have installed requirements for *development profile* you can manually install pep8 command-line tool.

```
$ pip install pep8
```

Then run it on a file or series of files to get a report of any violations.

```
$ pep8 --ignore=E501  apps/core/views.

apps/core/views.py:3:1: E302 expected 2 blank lines, found 1
apps/core/views.py:5:15: W291 trailing whitespace
apps/core/views.py:6:1: W293 blank line contains whitespace
apps/core/views.py:8:5: E101 indentation contains mixed spaces and tabs
apps/core/views.py:8:5: W191 indentation contains tabs
apps/core/views.py:9:5: E101 indentation contains mixed spaces and tabs
apps/core/views.py:9:5: W191 indentation contains tabs
apps/core/views.py:10:5: E101 indentation contains mixed spaces and tabs
apps/core/views.py:10:5: W191 indentation contains tabs
apps/core/views.py:11:5: E101 indentation contains mixed spaces and tabs
apps/core/views.py:11:5: W191 indentation contains tabs
apps/core/views.py:13:5: E101 indentation contains mixed spaces and tabs
apps/core/views.py:13:5: W191 indentation contains tabs
```

Find out more about this command line tool: http://pep8.readthedocs.io/en/latest/.

2.3.3 Examples

Importing

Importing is separated in 4 blocks:

- System import
- Django import
- 3rd party import
- Local import

There should be empty line between each block. Imports should usually be on separate lines.

```
# First we import system modules
import os
import sys
import importlib

# Then we import Django specific modeuls
from django.http import HttpResponse
from django.template.defaultfilters import slugify

# After that we import 3rd party modules
from ebooklib import epub

# At the end we import Django app specific or library specific modules
from . import loader
from .base import BaseConverter
```

Documenting

We are using Sphinx for documenting Python code. We don't use special Sphinx directives for arguments because we want to have our documentation readable in the code also.

Between doc string and rest of the code it is allowed to have empty line to make code more readable.

Here is example for test function:

```
def test_function(first, second):
    """This is my test function for the documentation.

    This test function does nothing at all. It is here just for the test purposes.
    There are many test functions but this function is special.

    .. code-block:: python

       test_function(booktype.someClass(), 21)

    :Args:
      - first (:class:`booktype.some.Class`): First argument
      - second: Second argument which also does nothing in test function
```

(continues on next page)

(continued from previous page)

```
    :Returns:
      Returns list of elements. For example: [1, 2, 3, 4, 5]

    :Raises:
      KeyError: An error occurred while working with out arguments.
    """

    return [first, second, first + second]
```

Ignored variable

If you need to assign something but will not need that variable, use _:

```
name, _ = unpack_userinfo()
```

Non public methods in class

When you want to have semi private methods or instance variables use leading underscore in the name.:

```
class TestClass:
    def __init__(self):
        self._example_for_instance = 21
        self._initial_load()

    def _initial_load(self):
        do_something()
```

2.4 JavaScript Style

2.4.1 Style

There are many different JavaScript coding styles but we are using Idiomatic JavaScript style in Booktype. There is one exception, we do not use extra space inside parentheses.

2.4.2 How to check

We are using JSHint, a tool that helps to detech errors and potential problems in JavaScript code.

First you need to install it:

```
$ npm install jshint -g
```

We have prepared configuration file for our coding style in **scripts/jshintrc** file. You can either install it globally or specify path to the config file each time you execute jshint.

```
$ jshint --config Booktype/scripts/jshintrc toc.js

toc.js: line 61, col 44, A leading decimal point can be confused with a dot: '.6'.
toc.js: line 69, col 64, Strings must use singlequote.
```

(continues on next page)

(continued from previous page)

```
toc.js: line 77, col 49, Missing space after 'if'.
toc.js: line 85, col 82, Strings must use singlequote.
toc.js: line 85, col 102, Strings must use singlequote.
toc.js: line 85, col 103, Trailing whitespace.
toc.js: line 86, col 83, Strings must use singlequote.
toc.js: line 87, col 79, Strings must use singlequote.
toc.js: line 88, col 79, Strings must use singlequote.
toc.js: line 88, col 88, Strings must use singlequote.
toc.js: line 89, col 85, Strings must use singlequote.
toc.js: line 91, col 80, Missing space after 'function'.
```

Read more in the official documentation how to integrate this tool inside of your text editor or IDE - http://www.jshint.com/install/.

2.4.3 Examples

Spaces

Never mix spaces and tabs. Please check your editor is correctly configured to use whitespace instead of tab. For readability we recommend setting two spaces representing a real tab.

Good

```
if (condition) {
  doSomething();
}

while (condition) {
  iterating++;
}

var Chapter = Backbone.Model.extend({
  defaults: {
    chapterID: null,
    title: "",
    urlTitle: "",
    isSection: false,
    status: null
  }
});

this.refresh = function () {
  var $this = this,
    lst = win.booktype.editor.data.chapters.chapters;

  jquery.each(lst, function (i, item) {
    $this.refreshItem(item);
  });

  this._checkForEmptyTOC();
};

var DEFAULTS = {
```

(continues on next page)

(continued from previous page)

```
  panels: {
    'edit': 'win.booktype.editor.edit',
    'toc' : 'win.booktype.editor.toc',
    'media' : 'win.booktype.editor.media'
  },

  styles: {
    'style1': '/static/edit/css/style1.css',
    'style2': '/static/edit/css/style2.css',
    'style3': '/static/edit/css/style3.css'
  },

  tabs: {
    'icon_generator': function (tb) {
      var tl = '';

      if (!_.isUndefined(tb.title)) {
        if (tb.isLeft) {
          tl = 'rel="tooltip" data-placement="right" data-original-title="' + tb.
↪title + '"';
        } else {
          tl = 'rel="tooltip" data-placement="left" data-original-title="' + tb.title␣
↪+ '"';
        }
      }

      return '<a href="#" id="' + tb.tabID + '"' + tl + '><i class="' + tb.iconID + '
↪"></i></a>';
    }
  }
}
```

2.5 Development profile

Booktype comes with already predefined development profile and a list of requirements for it.

2.5.1 Lazy development setup

Our development setup should look as much as possible as the production setup but it is possible to do development with the minimalistic setup also.

Check the *deployment documentation* which external services and libraries you need to install and have running.

```
# Create Python Virtual Environment
$ virtualenv --distribute bk20
$ cd bk20
$ source  bin/activate

# Clone Booktype
$ git clone https://github.com/booktype/Booktype.git

# Install Python requirements for development profile
$ pip install -r Booktype/requirements/dev.txt
```

(continues on next page)

(continued from previous page)

```
# Create Booktype project with development profile using Sqlite3
$ ./Booktype/scripts/createbooktype -p dev -d sqlite bkdev
$ cd bkdev

# Initialise project
$ ./manage.py syncdb
$ ./manage.py migrate
$ ./manage.py update_permissions
$ ./manage.py update_default_roles

# Edit bkdev_site/settings/dev.py file if needed

# Collect static files
$ ./manage.py collectstatic

# Run built-in web server in one shell
$ ./manage.py runserver 0.0.0.0:8000

# Run workers in another shell
$ ./manage.py celeryd -E --autoreload
```

2.6 Booktype core testing

2.6.1 Setup

Booktype source comes with ready tests. To be able to run tests one must install required development dependencies (if already functional Booktype project does not exists).

```
$ virtualenv --distribute bkproject
$ source bkproject/bin/activate
$ pip install -r <PATH_TO_BOOKTYPE>/requirements/dev.txt
$ cd <PATH_TO_BOOKTYPE>/tests
```

2.6.2 Tests

This is for Unit tests and some Integration tests. Works with in memory Sqlite3 database. Does not require any external service to be running.

Discover runner is configured to use 'test_*.py' pattern to find available tests. Django settings file used for the tests is settings.py.

```
$ ./start_tests.sh
```

Configuration

Using *settings.py* file in *tests/* directory. Definition for URL is in *urls.py* file.

When to write them

Mainly when writing Unit Tests. Considering it uses Sqlite3 database it can also be used for some Integration tests which are working with database.

Example

Example lib/booktype/apps/core/tests/test_templatetags.py.

```
import mock

from django.test import TestCase

from ..templatetags import booktype_tags

class UsernameTest(TestCase):
    def test_anonymous(self):
        user = mock.Mock(username = 'Booktype')
        user.is_authenticated.return_value = False

        self.assertEquals(booktype_tags.username(user), "Anonymous")


    def test_no_firstname(self):
        user = mock.Mock(username='booktype', first_name='')
        user.is_authenticated.return_value = True

        self.assertEquals(booktype_tags.username(user), "booktype")
```

2.6.3 Functional Tests

This is for some Integration tests and all Functional tests . Works with in memory sqlite3 database out of the box but could/should be configured to work with PostgreSQL also. To run these tests you need to have running Redis, RabbitMQ, Celery workers. This also includes Selenium tests.

Discover runner is configured to use 'functest_*.py' pattern to find available tests. Django settings file used for the tests is func_settings.py.

```
$ ./start_functests.sh
```

Configuration

Using *func_settings.py* file in *tests/* directory. Definition for URL is in *urls.py* file.

When to write them

For writing Integration tests and Functional tests. Tests are configured to use all external services (Redis, RabitMQ) and can Selenium can be freely used with them.

Write tests which are testing background workers for book conversion, communication with Sputnik, what our Web Server is returning to us, are we generating correct web pages and etc.

Example

Example lib/sputnik/tests/functest_connect.py

```
import json

from django.test import TestCase
from django.core.urlresolvers import reverse
from django.core.exceptions import PermissionDenied
from django.contrib.auth.models import User

import sputnik

class ConnectTest(TestCase):
    ERROR_MESSAGE = {'messages': [], 'result': False, 'status': False}
    EMPTY_MESSAGE = {'messages': [], 'result': True, 'status': True}

    def setUp(self):
        self.dispatcher = reverse('sputnik_dispatcher')
        user = User.objects.create_user('booktype', 'booktype@booktype.pro', 'password
↪')

    def test_anon_get_connect(self):
        response = self.client.get(self.dispatcher)

        self.assertEquals(response.status_code, 200)
        self.assertEquals(response.content, json.dumps(self.ERROR_MESSAGE))

    def test_get_connect(self):
        self.client.login(username='booktype', password='password')
        response = self.client.get(self.dispatcher, follow=True)

        self.assertEquals(response.status_code, 200)
        self.assertEquals(response.content, json.dumps(self.ERROR_MESSAGE))
```

2.6.4 Rules when writing tests

Each test method tests one thing. A test method must be extremely narrow in what it tests. A single test should never assert the behavior of multiple views, models, forms or even multiple methods within a class.

For views when possible use the Request factory. The *django.tets.client.RequestFactory* provides a wat to generate a request instance that can be used as the first argument to any view. This provides a greater amount of isolation then the standard Django test client, but it does require a little bit of extra work.

```
from django.test import TestCase
from django.test.client import RequestFactory

class SimpleTest(TestCase):
    def test_details(self):
        factory = RequestFactory()

        request = factory.get('/customer/details')
```

Don't write tests that have to be tested.

No Fixtures. Working with fixtures can be problematic when we upgrade our models. New fixture data needs to be created plus all references to it (in the code and in the tests) must be updated. It is recommended to use tools for

generating test data like Factory Boy (https://pypi.python.org/pypi/factory_boy/).

Use mock objects. In our Unit Tests we should use as much as possible mock objects (if it is possible). For that we use Python Mocking and Patching Library for Testing (https://pypi.python.org/pypi/mock).

2.6.5 More info

- https://docs.djangoproject.com/en/dev/topics/testing/
- Tools
  - https://pypi.python.org/pypi/mock - A Python Mocking and Patching Library for Testing
  - https://pypi.python.org/pypi/factory_boy/ - A verstile test fixtures replacement based on thoughtbot's factory_girl for Ruby
  - https://pypi.python.org/pypi/django-discover-runner - A Django test runner based on unittest2's test discovery
- Books
  - http://chimera.labs.oreilly.com/books/1234000000754 - Test-Driven Development with Python
  - http://www.packtpub.com/python-testing-cookbook/book - Python Testing Cookbook
  - http://www.amazon.com/Python-Testing-Beginners-Daniel-Arbuckle/dp/1847198848 - Python Testing: Beginner's Guide
  - http://gettingstartedwithdjango.com/ - Getting Started with Django
  - http://effectivedjango.com/testing.html - Testing in Django
  - http://www.obeythetestinggoat.com/ - Objey the Testing Goat!

2.7 Documenting

Sphinx is used for documenting Booktype. The idea is to make our documentation readable (especially docstrings) in raw format. Developers looking at the documentation will spend their time in the editor. They do not want to look at cryptic text formatting which is only understandable after it has been converted to HTML. Having clean docstrings we make our documentation accessible both in HTML and raw format.

2.7.1 Markup

This document is not going to teach you how to use Sphinx. For that we recommend looking at First Steps and Sphinx documentation.

Here are couple of examples how you might want to format you Python Docstrings.

Arguments and return value

Between doc string and rest of the code it is allowed to have empty line to make code more readable.

```
def test_function(first, second):
    """This is my test function for the documentation.

    This test function does nothing at all. It is here just for the test purposes.
```

(continues on next page)

(continued from previous page)

```
    There are many test functions but this function is special.

    :Args:
      - first (:class:`booktype.some.Class`): First argument
      - second: Second argument which also does nothing in test function

    :Returns:
      Returns list of elements. For example: [1, 2, 3, 4, 5]

    :Raises:
      KeyError: An error occurred while working with out arguments.
    """

    return [first, second, first + second]
```

Include code

It is recommened to include code samples which could clarify usage of code we are trying to document.

```
def sample_function(a, b):
    """My sample function.

    .. code-block:: python

       a = 4 + 3
       ret = sample_function(a, 3)
       print ret
    """
```

Note

There is something we want to emphasize? Use notes. It will nicely wrap your text in a block.

```
def sample_function(a, b):
    """My sample function.

    .. note::

       Please note this is only a sample function.
    """
```

Warnings

There might be some issues when using this code? It could be dangerous if not used properly? Warned the user in that case! Your warning message will be wrapped in a block easily noticeable by user.

```
def sample_function(a, b):
    """My sample function.

    .. warning::

       This function is not working.
    """
```

Versions

At some point we will have to modify or break our API. This we need to document.

```
def sample_function(a, b):
    """My sample function.

       .. versionadded:: 1.5
          We added new b argument.

       .. versionchanged:: 2.0
          Argument b can be also string.

       .. deprecated:: 3.1
          Use :func:`sample_function_new` instead.
    """
```

Reference other code

Referencing other functions.

```
def sample_function(a, b):
    """My sample function.

       This function is similar to :func:`sample_function_new`.
    """
```

Referencing other classses or modules.

```
class SampleClass:
    """My sample class.

       This class is similar to :class:`booktype.sample.SampleClassNew`.
       Module :mod:`booktype.contrib.sample` is used for
    """
```

Sometimes we need to include a list of references or external documents. These lists are created using the seealso directive:

```
def sample_function(a, b):
    """My sample function.

       .. seealso::

          `GNU sample manual, Basic Sample Format <http://link>`_ Documentation for␣
↪Sample Format.
    """
```

2.7.2 Building

Note: Before doing these steps it is recommended to define environment variables **PYTHONPATH** and **DJANGO_SETTINGS_MODULE**.

```
$ cd docs
$ make html
```

Documentation will be in directory *_build/html/*.

CHAPTER 3

Deployment

For installation and usage instructions, please read the manual Booktype for Authors and Publishers.

- **Project:** *Structure* | *Profiles*
- **Settings:** *Booktype* | *PostgreSQL* | *Redis* | Supervisor | RabbitMQ
- **Publishing:** *MOBI*
- **Scripts:** *Createbooktype* | Backup
- **Upgrades:** *Instructions*
- **Unoffical platforms:** *CentOS* | *Mac OS X* | *Nginx*

CHAPTER 4

Development

Learn about the development and testing of Booktype:

- **Contributing:** *How to contribute* | *Before you send a pull request*
- **Setup:** *Development profile*
- **Code:** *Python style* | *JavaScript style* | *Testing* | *Documenting*

CHAPTER 5

Community

- **Forum:** Support forum | Development forum | Documentation forum
- **Resources:** GitHub | Tracker | Translation platform

CHAPTER 6

Indices and tables

- genindex
- search

Index

B

BOOKTYPE_CONVERTER_MODULES
    setting, 7
BOOKTYPE_ROOT
    setting, 6
BOOKTYPE_SITE_DIR
    setting, 6
BOOKTYPE_SITE_NAME
    setting, 6
BOOKTYPE_URL
    setting, 6

C

COMPRESS_ENABLED
    setting, 8
COMPRESS_OFFLINE
    setting, 8
COVER_IMAGE_UPLOAD_DIR
    setting, 7

D

DATA_ROOT
    setting, 7
DATA_URL
    setting, 7
DEPRECATED
    setting, 8

P

PROFILE_ACTIVE
    setting, 6
PROFILE_IMAGE_UPLOAD_DIR
    setting, 7

R

REDIS
    setting, 7
REDIS_DB
    setting, 8
REDIS_HOST
    setting, 7
REDIS_PASSWORD
    setting, 8
REDIS_PORT
    setting, 8

S

setting
    BOOKTYPE_CONVERTER_MODULES, 7
    BOOKTYPE_ROOT, 6
    BOOKTYPE_SITE_DIR, 6
    BOOKTYPE_SITE_NAME, 6
    BOOKTYPE_URL, 6
    COMPRESS_ENABLED, 8
    COMPRESS_OFFLINE, 8
    COVER_IMAGE_UPLOAD_DIR, 7
    DATA_ROOT, 7
    DATA_URL, 7
    DEPRECATED, 8
    PROFILE_ACTIVE, 6
    PROFILE_IMAGE_UPLOAD_DIR, 7
    REDIS, 7
    REDIS_DB, 8
    REDIS_HOST, 7
    REDIS_PASSWORD, 8
    REDIS_PORT, 8