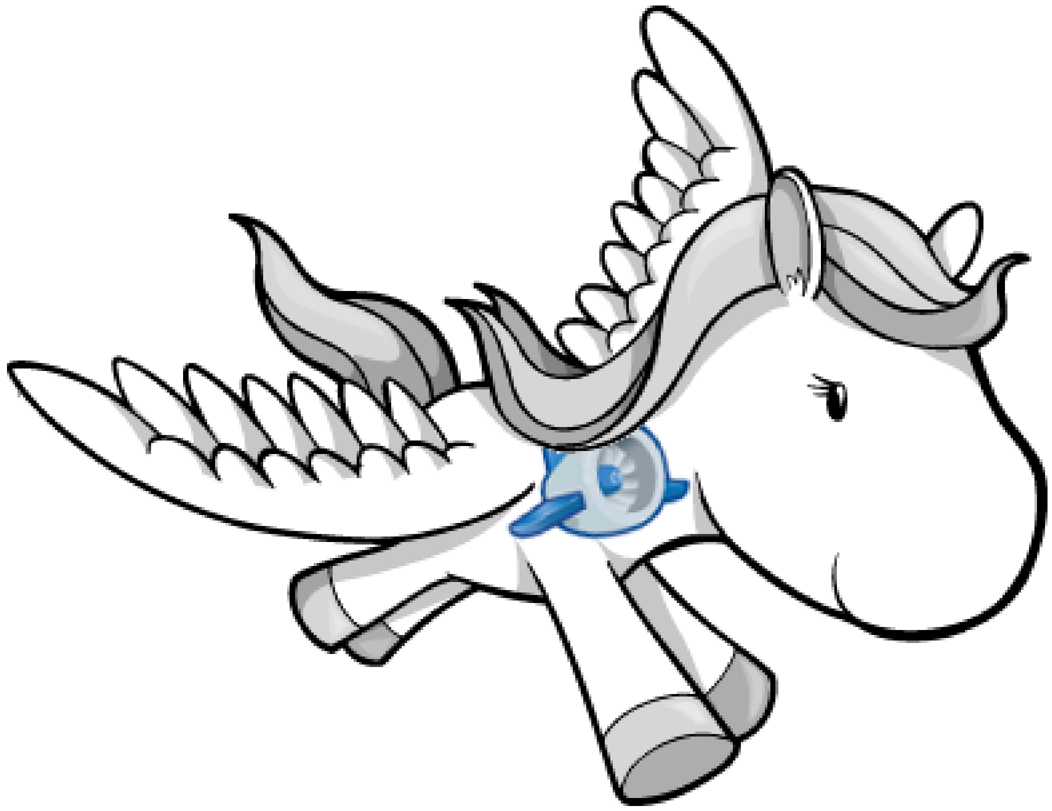
Django on AppEngine Documentation

Release

Sebastian Pawluś

December 06, 2016

Contents



Welcome to the documentation for django-rocket-engine - Django on AppEngine

Project is a helper library with main goal to help setup "correct" Django development environment for Google AppEngine services.

Supports:

- basic support for pip style requirements,
- support for virtualenv environments,
- pre/post deployment hooks,
- seamless BlobStorage backend.

The project is still in the experimental stage, being more like proof-of-concept. Source code can be found on github.

This project was inspired by the work of Waldemar Kornewald and Thomas Wanschik from All Buttons Pressed, some ideas where moved from djangoappengine project.

Example application

CHAPTER 1

Installation

1.1 Download latest Google AppEngine SDK

Get the latest version of SDK, if you are using Linux please make sure that SDK is available on your PATH (how?).

1.2 Install Django

Install Django framework. There are many ways of doing that (suggestion is to use virtualenv along with virtualenvwrapper and pip)

```
$ pip install django==1.3.1
```

Note: Version 1.3.1 is latest supported by SDK

1.3 Create Django project

Create a new, awesome Django project. Right now it's even more awesome because this project will use AppEngine:

```
$ django-admin.py startproject my_awesome_project
```

1.4 Install django-rocket-engine

Install latest version of django-rocket-engine, with pip

```
$ pip install django-rocket-engine
```

1.5 Register application on Google AppEngine

Register new application on Google AppEngine site.

1.6 Create CloudSQL database

Create a CloudSQL database instance using Google Api Console, add application instance name from previous step in "Authorized applications". Using "SQL Prompt" tab create a database inside your CloudSQL instance.

```
CREATE DATABASE database_name;
```

1.7 Configuration

Google AppEngine requires applications to have an config in app.yaml file, which is responsible for basic description, and how to manage the application. Create app.yaml inside project directory. Example of app.yaml for project.

```
# app.yaml
application: unique_appengine_appspot_id
version: 1
runtime: python27
api_version: 1
threadsafe: true

handlers:
- url: /.*
  script: rocket_engine.wsgi

libraries:
- name: django
  version: 1.3

env_variables:
  DJANGO_SETTINGS_MODULE: 'settings'
```

Things that need to be done in settings.py are presented in code snippet below:

```
# settings.py
from rocket_engine import on_appengine

...

# django-rocket-engine as every Django application
# needs to be added to settings.py file in INSTALLED_APPS section:
INSTALLED_APPS = (
    # other django applications here
    # ...

    'rocket_engine',
)


# remove project name from ROOT_URLCONF.
# AppEngine doesn't treat project as a module
# like normal Django application does.
ROOT_URLCONF = 'urls'

# to use different databases  during
# development process and on production.
if on_appengine:
    DATABASES = {
        'default': {
            'ENGINE': 'rocket_engine.db.backends.cloudsql',
            'INSTANCE': 'instance:name',
            'NAME': 'database_name',
        }
    }
else:
    DATABASES = {
        'default': {
            'ENGINE': 'django.db.backends.sqlite3',
            'NAME': 'development.db'
        }
    }

# disable debugging on production
DEBUG = not on_appengine
```

Note: Instead of using sqlite3 backend your are able to use MySQL backend. This should also be your choice for serious applications.

That's just about it. Application is ready to run:

```
$ python manage.py syncdb
$ python manage.py runserver
```

and deploy:

```
$ python manage.py on_appengine syncdb
$ python manage.py appengine update --oauth2
```

Have fun!

1.7.1 Next Steps

Requirements

Google AppEngine SDK libraries:

Google AppEngine SDK comes with sets of libraries. If there is a need to use one of them, you should append the library in your libraries seduction in app.yaml file rather to add them with use of requirements.txt file (explained below). Example of how to enable lxml in application.

```
# app.yaml

# ...

libraries:
- name: django
  version: 1.3
- name: lxml
  version: 2.3
```

Python requirements.txt

To bring AppEngine development to more pythonic status. The library comes with basic support for python packaging system, You can keep list of required packages in requirements.txt file in your project root directory. Example of requirements.txt for simple project may contains packages like:

```
django-tastypie
django-taggit>=0.4
```

These packages will be downloaded and installed during deployment stage (manage.py appengine update). Requirements file may also contain references to packages being under source control:

```
git+git://github.com/alex/django-taggit.git
git+git://github.com/jezdez/django-dbtemplates.git@master
```

Note: There is no need to add django or django-rocket-engine to your requirements.txt file. Those requirements are already satisfied.

Note: Editable requirements (prepended with -e option) are not supported.

More about using requirements file might be read here.

Commands

appengine

Google AppEngine comes with appcfg.py to cover all functionality of the deployment process. This command is currently now wrapped by appengine django command and comes with some benefits of configuration hooks:

```
python manage.py appengine update
```

Calling this command will send your code on remote AppEngine instance. This option comes with support of pre and post update hooks see Settings.

on_appengine

To perform an operation on Google AppEngine from your local machine use:

```
python manage.py on_appengine syncdb
```

This command will perform a sycndb operation on your remote instance. Google AppEngine doesn't come with any kind of remote access mechanism (like SSH, Talent), this command helps to overcome this inconvenience. Any command invoked this way will be called to use of remote storage instead of your local one. This command only affects storage. Other useful examples might be.

- remote python shell:

```
    python manage.py on_appengine shell
```

- remote CloudSQL shell:

```
python manage.py on_appengine dbshell
```

- migrate database if South is being used:

```
python manage.py on_appengine migrate
```

Blob Storage

To enable BlobStorage system as a Django storage, modify your code with elements presented below.

```
# urls.py
urlpatterns = patterns(
    ...
    url(r'^media/(?P<filename>.*)/$','rocket_engine.views.file_serve'),
)
```

```
# settings.py
DEFAULT_FILE_STORAGE = 'rocket_engine.storage.BlobStorage'
```

Settings

DATABASES

django-rocket-engine comes with pre-defined backend Google CloudSQL wrapper which prevents using your production database during development:

```
DATABASES = {
    'default': {
        'ENGINE': 'rocket_engine.db.backends.cloudsql',
        'INSTANCE': 'instance:name',
        'NAME': 'database_name',
    }
}
```

To distinguish between production and development. The library provides helper method which could applied in settings.py:

```
# settings.py
from rocket_engine import on_appengine

...

if on_appengine:
    DATABASES = {
        'default': {
            'ENGINE': 'rocket_engine.db.backends.cloudsql',
            'INSTANCE': 'instance:name',
            'NAME': 'database_name',
        }
    }
else:
    DATABASES = {
        'default': {
            'ENGINE': 'django.db.backends.sqlite3',
            'NAME': 'development.db'
        }
    }
```

DEFAULT_FILE_STORAGE

Use this setting to setup Blob objects as a default project storage.

> DEFAULT_FILE_STORAGE = 'rocket_engine.storage.BlobStorage'

APPENGINE_PRE_UPDATE

Callable that will be applied before sending application to Google AppEngine.

Default:

```
APPENGINE_PRE_UPDATE = 'appengine_hooks.pre_update'
```

APPENGINE_POST_UPDATE

Callable that will be applied after sending application to Google AppEngine.

Default:

```
APPENGINE_POST_UPDATE = 'appengine_hooks.post_update'
```

Example of appengine_hooks.py file:

```
from django.core.management import call_command

def pre_update():
    call_command('collectstatic')

def post_update():
    call_command('on_appengine', 'syncdb')

    # If south is being used
    call_command('on_appengine', 'migrate')
```