django_spanner Documentation

Release 0.0.2

micropyramid

February 13, 2017

Contents

Django Spanner is a simple package used to deploy and manage Django applications. Django Spanner is a set of commands used to deploy and manage django projects. It internally uses Fabric. This can also be called as **Deploying Django with Fabric**.

- You can use this to deploy & manage any django application with just few configurations. Configure once and run many times.
- As well as you can deploy & manage single django application on multiple remote servers (multiple staging servers, multiple live servers). You can specify different configuration for each type of server (stage/live) like here sample config file
- Also used to manage local django project to install requirements, run make migrations, migrate, take database backups and many more...

This package is developed by MicroPyramid team. Please refer the github repository for the django-spanner source code. It's free and open source.

CHAPTER 1

Github Repository

Django Spanner - https://github.com/MicroPyramid/django-spanner

1.1 Contents

1.1.1 Features

- Install requirements in a virtualenv.
- Migrate database.
- Execute any management command.
- Rsync files to destination server(stage/live) with and without settings file.
- Deploy the django application to server(Installs requirements, migrates database, rsync files, runs uwsgi server).
- Take database backups(server backups as well as local).
- Restore local and server databases.
- Reset local and server databases.
- Restart server, celery, supervisor, uwsgi.
- Rebuild index, collect static.

Note: You can execute all these commands in local as well as remote servers.

1.1.2 Installation

The recommended way to install the django-spanner into a virtualenv using pip:

```
pip install django-spanner
```

Or, install using the latest version from GitHub:

```
git clone https://github.com/MicroPyramid/django-spanner.git

cd django_spanner

python setup.py install
```

1.1.3 Setup

- First, create an YAML file similar to sample_config.yaml and fill the configuration details.
- Next, create a file named `fabfile.py` in your project directory and import all functions(fab commands/tasks) from *django_spanner*.
- Finally, call the `setup()` function with your configuration yaml file path.

Here is an example fabfile -

```
# fabfile.py
from django_spanner.commands import *
setup("config_file_name.yaml")
```

1.1.4 Commands

Usage

```
    fab <run_local/run_stage/run_live> <command_name>
```

List Commands - shows the list of all available fab commands

```
        fab -l
```

Install Requirements

- To install the requirements on your local system:

```
        fab run_local activate_env_install_requirements

                (or)

        fab activate_env_install_requirements
```

- To install the requirements on your remote staging servers:

```
        fab run_stage activate_env_install_requirements
```

- To install the requirements on your remote live servers:

```
        fab run_live activate_env_install_requirements
```

Rsync project to remote server(stage/live)

To rsync project local files to remote destination server -

- with settings file -

```
        fab <run_stage/run_live> rsync_with_settings
```

- without settings file -

```
        fab <run_stage/run_live> rsync_without_settings
```

Deploy To Server

This commands copy local project files to destination(stage/live) servers, installs requirements, applies migrations and finally runs uWSGI server(both in debug and deployment modes)

```
fab <run_stage/run_live> deploy_to_server
```

By default, this command rsyncs project files without settings file and runs touch command for project uwsgi file under /etc/uwsgi/vassals/ folder.

- To rsync with settings file and to run uwsgi in debug mode:

```
fab <run_stage/run_live> deploy_to_server:sync_with_setting='true',debug='true'
```

Note: It automatically creates project_root, env in server if not exists

Local database backup

```
fab take_local_backup
```

Server database backup

```
fab <run_stage/run_live> take_server_backup
```

Restore Server database to Local

```
fab <run_stage/run_live> take_server_backup
fab restore_to_local
```

Reset Local database

```
fab reset_local_db
```

Reset Server database

```
fab <run_stage/run_live> reset_server_db
```

Run Management Commands

This function is used to run management commands -

```
fab <run_local/run_stage/run_live> manage_py:<management_command_name>
```

- To apply migrations

```
fab <run_local/run_stage/run_live> migrate
```

- Execute collect static

```
fab <run_local/run_stage/run_live> collect_static
```

- Rebuild search index

```
fab <run_local/run_stage/run_live> rebuild_index
```

- To restart celery in remote servers

```
fab <run_stage/run_live> restart_celery
```

- To restart supervisorctl in remote servers

```
fab <run_stage/run_live> restart_supervisior
```

- To restart uwsgi in remote servers

```
fab <run_stage/run_live> restart_uwsgi
```

- To restart remote servers

```
fab <run_stage/run_live> restart_server
```