SatNOGS DB

Release 0+untagged.50.gc1b164d.dirty

SatNOGS

Apr 05, 2024

CONTENTS

CHAPTER

ONE

INSTALLATION

1.1 Docker Installation

1. **Requirements**

   You will need docker and docker-compose.

2. **Get the source code**

   Clone source code from the repository:

```
$ git clone https://gitlab.com/librespacefoundation/satnogs/satnogs-db.git
$ cd satnogs-db
```

3. **Configure settings**

   Set your environmental variables:

```
$ cp env-dist .env
```

4. **Install frontend dependencies**

   Install dependencies with `npm`:

```
$ npm install
```

   Test and copy the newly downlodaded static assets:

```
$ ./node_modules/.bin/gulp
```

5. **Run it!**

   Run satnogs-db:

```
$ docker-compose up -d --build
```

6. **Populate database**

   Create, setup and populate the database with demo data:

```
$ docker-compose exec web djangoctl.sh initialize
```

   Your satnogs-db development instance is available in localhost:8000. Go hack!

1.2 VirtualEnv Installation

1. **Requirements**

   You will need python, python-virtualenvwrapper, pip and git

2. **Get the source code**

   Clone source code from the repository:

```
$ git clone https://gitlab.com/librespacefoundation/satnogs/satnogs-db.git
$ cd satnogs-db
```

3. **Build the environment**

   Set up the virtual environment. On first run you should create it and link it to your project path.:

```
$ mkvirtualenv satnogs-db -a .
```

4. **Configure settings**

   Set your environmental variables:

```
$ cp env-dist .env
```

5. **Install frontend dependencies**

   Install dependencies with `npm`:

```
$ npm install
```

   Test and copy the newly downlodaded static assets:

```
$ ./node_modules/.bin/gulp
```

6. **Run it!**

   Activate your python virtual environment:

```
$ workon satnogs-db
```

   Just run it:

```
(satnogs-db)$ ./bin/djangoctl.sh develop .
```

7. **Populate database**

   Create, setup and populate the database with demo data:

```
(satnogs-db)$ ./bin/djangoctl.sh initialize
```

   Your satnogs-db development instance is available in localhost:8000. Go hack!

CHAPTER

TWO

DEVELOPER GUIDE

Thank you for your interest in developing SatNOGS! There are always bugs to file; bugs to fix in code; improvements to be made to the documentation; and more.

The below instructions are for software developers who want to work on satnogs-db code.

2.1 Workflow

When you want to start developing for SatNOGS, you should *follow the installation instructions*, then…

1. Read CONTRIBUTING.md file carefully.
2. Fork the upstream repository in GitLab.
3. Code!
4. Test the changes and fix any errors by running tox.
5. Commit changes to the code!
6. When you're done, push your changes to your fork.
7. Issue a merge request on Gitlab.
8. Wait to hear from one of the core developers.

If you're asked to change your commit message or code, you can amend or rebase and then force push.

If you need more Git expertise, a good resource is the Git book.

2.2 Templates

satnogs-db uses Django's template engine templates.

2.3 Frontend development

Third-party static assets are not included in this repository. The frontend dependencies are managed with `npm`. Development tasks like the copying of assets, code linting and tests are managed with `gulp`.

To download third-party static assets:

1. Install dependencies with `npm`:

```
$ npm install
```

2. Test and copy the newly downlodaded static assets:

```
$ ./node_modules/.bin/gulp
```

To add new or remove existing third-party static assets:

1. Install a new dependency:

```
$ npm install <package>
```

2. Uninstall an existing dependency:

```
$ npm uninstall <package>
```

3. Copy the newly downlodaded static assets:

```
$ ./node_modules/.bin/gulp assets
```

2.4 Dependency Management

Dependencies of the package are defined in the `setup.cfg` file. Only top-level dependencies shall be defined, with the exception of overrides needed to workaround dependency incompatibilities. Each dependency is defined using the appropriate compatible version specifier. The compatible version specifiers shall be such that any newer compatible version can be installed, taking into account the versioning schema followed by each dependency. The `dev` extra contains packages and tools required for development, testing and packaging. When a dependency has been added, removed or updated in `setup.cfg`, the requirement files must be manually regenerated. To regenerate these files run:

```
$ ./contrib/refresh-requirements-docker.sh
```

The script will update the following files:

- `requirements.txt` - List of resolved dependencies for the package
- `requirements-dev.txt` - List of resolved dependencies for the development tools
- `constraints.txt` - List of resolved dependency constraints for both the package and development tools

Changes on the above files shall be committed together with the changes in `setup.cfg`.

2.5 Documentation

The documentation can be generated locally with sphinx:

```
$ cd docs
$ virtualenv -p python3 env
$ source env/bin/activate
$ pip install sphinx_rtd_theme
$ make html SPHINXOPTS="-W"
```

2.6 Coding Style

Follow the PEP8 and PEP257 Style Guides.

2.7 What to work on

You can check open issues. We regurarly open issues for tracking new features. You pick one and start coding.

CHAPTER

THREE

MAINTENANCE

3.1 Updating Python dependencies

To update the Python dependencies:

1. Execute script to refresh `requirements{-dev}.txt` files:

```
$ ./contrib/refresh-requirements.sh
```

2. Stage and commit `requirements{-dev}.txt` files.

3.2 Updating frontend dependencies

The frontend dependencies are managed with `npm`. To update the frontend dependencies, while respecting semver:

1. Update all the packages listed in `package.json`:

```
$ npm update
```

2. Test and copy the newly downlodaded static assets:

```
$ ./node_modules/.bin/gulp
```

3. Stage and commit `package-lock.json` file.

CHAPTER

FOUR

RELEASING

4.1 Versioning scheme

This repository follows PEP-440 versioning scheme. All releases must use a *X.Y* segment version which signifies a final project release and is compatible with Semantic Versioning. The versions must be numbered in a consistently increasing fashion. Major *X* will never need to be increased unless the application is completely rewritten. Minor *Y* shall be increased on each release. A Patch or additional segments, as described in SemVer, shall not be used.

4.2 Release procedure

To make a new release:

1. Find the next available minor version among the whole set of already present tags in the repository.
2. Create an annotated tag from *master* branch in GitLab with a commit message:

```
Tag version 'X.Y'
```

CHAPTER

FIVE

API

SatNOGS DB API is a REST API that provides detailed information about Satellites and Transmitters. This document explains how to use the API to retrieve data for your application.

5.1 Using API Data

API access is open to anyone. All API data are freely distributed under the CC BY-SA license.

5.2 API Reference

Our live schema docs contain a full interactive reference of the API.