satnogs-network Documentation

Release 1

SatNOGS

Feb 18, 2020

Contents

1 Installation 1

2 Developer Guide 5

3 Releasing 9

4 Maintenance 11

CHAPTER 1

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-network.git
$ cd satnogs-network
```

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-network:

```
$ 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-network development instance is available in localhost:8000. Go hack!

7. **Clean database**

   Clean up the database in case of problems:

```
$ docker-compose exec web django-admin flush
```

8. **Build the documentation locally**

   $ tox -e docs

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-network.git
$ cd satnogs-network
```

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-network -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-network
```

   Just run it:

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

7. **Populate database**

   Create, setup and populate the database with demo data:

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

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

CHAPTER 2

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-network 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 by *Running the tests locally* and fix any errors.
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-network 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 Simulating station heartbeats

Only stations which have been seen by the server in the last hour (by default, can be customized by *STATION_HEARTBEAT_TIME*) are taken into consideration when scheduling observations. In order to simulate an heartbeat of the stations 7, 23 and 42, the following command can be used:

```
docker-compose exec web django-admin update_station_last_seen 7 23 42
```

2.5 Manually run a celery tasks

The following procedure can be used to manually run celery tasks in the local development environment:

1. *Install the docker-based development environment*.
2. Start a django-admin shell:

```
docker-compose exec web django-admin shell
```

3. Run an asnyc task and check if it succeeded:

```
from network.base.tasks import update_all_tle
task = update_all_tle.delay()
assert(task.ready())
```

4. (optional) Check the celery log for the task output:

```
docker-compose logs celery
```

2.6 Running the tests locally

To test your changes to the code locally with tox in the same way the CI does you can follow these steps:

1. Setup a new virtual environment (this shouldn't be the same virtual environment you might have created for the *VirtualEnv Installation*):

```
mkvirtualenv network-test -a .
```

2. Install tox in the same version defined by `GITLAB_CI_PYPI_TOX` in .gitlab-ci.yml:

```
pip install tox~=3.8.0
```

3. Run the tests:

```
tox -e "flake8,isort,yapf,pylint"
```

2.7 Coding Style

Follow the PEP8 and PEP257 Style Guides.

2.8 What to work on

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

CHAPTER 3

Releasing

3.1 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.

3.2 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 4

Maintenance

4.1 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.

4.2 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.