SatNOGS Network

Release 1.113+0.g284910d.dirty

SatNOGS

Apr 04, 2024

CONTENTS

CHAPTER

ONE

INSTALLATION

1.1 Requirements and info

Python: CPython 3.6+

Operating system: Linux

Prerequisites: docker-compose (recommended) or virtualenv, npm.

Git repository: https://gitlab.com/librespacefoundation/satnogs/satnogs-network.git

1.2 Clone

Consult the GitLab page on how to clone the repository.

1.3 Configuration

Set your environmental variables:

```
$ cp env-dist .env
$ ${EDITOR} .env
```

1.4 Installation

1.4.1 Quick install

The recommended quick install method is to use **Docker Compose**. This method will start separate containers for Django runserver, Celery, MariaDB and Redis.

The **Virtualenv** quick install method lacks a task queue, uses SQLite as a database and memcached for caching. It is only recommended for frontend development.

Docker Compose (recommended)

Change into the cloned repository directory:

```
$ cd <path/to/clone>
```

To run SatNOGS Network service:

```
$ ./satnogs.sh up
```

To stop SatNOGS Network service:

```
$ ./satnogs.sh compose down
```

To clean-up SatNOGS Network installation:

```
$ ./satnogs.sh clean
```

Virtualenv

This installation method is recommended only for frontend development purposeses.

Change into cloned repository directory:

```
$ cd <path/to/clone>
```

To run SatNOGS Network in a virtualenv:

```
$ ./satnogs.sh develop
```

To remove SatNOGS Network virtualenv:

```
$ ./satnogs.sh remove
```

1.4.2 Production install

For production installations, check Deploying Django.

Gunicorn

If `gunicorn` is used as the WSGI server then to start the application use:

```
$ ./bin/djangoctl.sh run
```

and to bring Celery up:

```
$ ./bin/djangoctl.sh run_celery
```

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-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 Backend development

When running satnogs-network using the docker container the webserver auto-reloads when files get changed. You need to restart the network-web container only when you change something in *settings.py*. All the other changes are directly applied with refreshing the page you are currently working on.

2.5 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
```

2.6 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
```

2.7 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"
```

2.8 Coding Style

Follow the PEP8 and PEP257 Style Guides.

2.9 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 Network API is a REST API to get scheduled jobs and post observation data. This document explains how to use the API to retrieve and post 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

The SatNOGS Network API is described by the SatNOGS Network OpenAPI document which follows the OpenAPI specification.

5.3 Interactive API documentation

Based on the OpenAPI document for SatNOGS Network, an interactive API documentation is available, auto-generated by the OpenAPI generator.