School Navigator Documentation

Release 0.1

Code for Durham

December 02, 2015

Durham School Navigator is a website to help Durham, NC residents navigate the Durham Public School System. The open source project is built and maintained by Code for Durham.

The official site is online at https://schools.codefordurham.com/. The code is available on GitHub. The email list is on Google Groups.

The following documentation outlines how to get started developing code for the project.

Contents:

CHAPTER 1

Development Setup

Follow these steps to contribute to the School Navigator project!

1.1 Project Architecture

The project is divided into two components, the frontend and backend:

- *Frontend*: front facing user interface. Built in AngularJS, HTML, and CSS.
- *Backend*: backend API that powers the frontend. Built in Python/Django, PostgreSQL, and PostGIS.

Generally, unless you're working on the REST API, you'll develop the *Frontend*.

1.2 Clone the Repository

To get started, you'll first need to clone the GitHub repository so you can work on the project locally. In a terminal, run:

```
git clone git@github.com:codefordurham/school-navigator.git
cd school-navigator
```

1.3 Frontend Setup

Once you've cloned the project, open the `frontend` directory:

```
cd frontend/
```

Next run a basic HTTP server with Python:

```
# Python <= 2.7
python -m SimpleHTTPServer
# Python >= 3.0
python -m http.server
```

Now visit http://localhost:8000/ in your browser.

1.4 Backend Setup

Below you will find basic setup instructions for the school_inspector project. To begin you should have the following applications installed on your local development system:

- Python >= 3.4 (3.4 recommended)
- pip >= 1.5
- virtualenv >= 1.11
- virtualenvwrapper >= 3.0
- Postgres >= 9.1
- git >= 1.7

The deployment uses SSH with agent forwarding so you'll need to enable agent forwarding if it is not already by adding `ForwardAgent yes` to your SSH config.

1.4.1 Getting Started

Note: The following instructions use the apt package manager for Debian/Ubuntu Linux. If apt is not available for your system, use your preferred package manager (i.e. Homebrew for Mac OS X) to install the required dependencies.

If you need Python 3.4 installed, you can use this PPA:

```
sudo add-apt-repository ppa:fkrull/deadsnakes
sudo apt-get update
sudo apt-get install python3.4-dev
```

The tool that we use to deploy code is called Fabric, which is not yet Python3 compatible. So, we need to install that globally in our Python2 environment:

```
sudo pip install fabric==1.8.1
```

To setup your local environment you should create a virtualenv and install the necessary requirements:

```
# Check that you have python3.4 installed
$ which python3.4
$ mkvirtualenv school_navigator -p `which python3.4`
(school_navigator)$ $VIRTUAL_ENV/bin/pip install -r $PWD/requirements/dev.txt
```

Then create a local settings file and set your `DJANGO_SETTINGS_MODULE` to use it:

```
(school_navigator)$ cp school_navigator/settings/local.example.py school_navigator/settings/local.py
(school_navigator)$ echo "DJANGO_SETTINGS_MODULE=school_navigator.settings.local" > .env
```

Exit the virtualenv and reactivate it to activate the settings just changed:

```
deactivate
workon school_inspector
```

If you're on Ubuntu 12.04, to get get postgis you need to set up a few more packages before you can create the db and set up the postgis extension:

```
sudo apt-add-repository ppa:ubuntugis/ppa
sudo aptitude update && sudo aptitude install postgis postgresql-9.1-postgis-2.0 postgresql-9.1-postg
```

Now, create the Postgres database and run the initial syncdb/migrate:

```
(school_navigator)$ createdb -E UTF-8 school_navigator
(school_navigator)$ python manage.py migrate
```

You should now be able to run the development server:

```
python manage.py runserver
```

CHAPTER 2

Server Setup

2.1 Provisioning

The server provisioning is managed using Salt Stack. The base states are managed in a common repo and additional states specific to this project are contained within the `conf` directory at the root of the repository.

For more information see the provisioning guide.

2.2 Layout

Below is the server layout created by this provisioning process:

```
/var/www/school_navigator/
    source/
    env/
    log/
    public/
        static/
        media/
    ssl/
```

`source` contains the source code of the project. `env` is the virtualenv for Python requirements. `log` stores the Nginx, Gunicorn and other logs used by the project. `public` holds the static resources (css/js) for the project and the uploaded user media. `public/static/` and `public/media/` map to the `STATIC_ROOT` and `MEDIA_ROOT` settings. `ssl` contains the SSL key and certificate pair.

2.3 Deployment

For deployment, each developer connects to the Salt master as their own user. Each developer has SSH access via their public key. These users are created/managed by the Salt provisioning. The deployment itself is automated with Fabric. To deploy, a developer simply runs:

```
# Deploy updates to staging
fab staging deploy
# Deploy updates to production
fab production deploy
```

This runs the Salt highstate for the given environment. This handles both the configuration of the server as well as updating the latest source code. This can take a few minutes and does not produce any output while it is running. Once it has finished the output should be checked for errors.

Server Provisioning

3.1 Overview

School_Navigator is deployed on the following stack.

- OS: Ubuntu 14.04 LTS
- Python: 3.4
- Database: Postgres 9.3
- Application Server: Gunicorn
- Frontend Server: Nginx
- Cache: Memcached

These services can configured to run together on a single machine or on different machines. Supervisord manages the application server process.

3.2 Salt Master

Each project needs a Salt Master per environment (staging, production, etc). The master is configured with Fabric. `env.master` should be set to the IP of this server in the environment where it will be used:

```
@task
def staging():
    ...
    env.master = <ip-of-master>
```

You will need to be able to connect to the server as a root user. How this is done will depend on where the server is hosted. VPS providers such as Linode will give you a username/password combination. Amazon's EC2 uses a private key. These credentials will be passed as command line arguments.:

```
# Template of the command
fab -u <root-user> <environment> setup_master
# Example of provisioning a Linode VM for staging
fab -u root staging setup_master
# Example of provisioning an AWS VM for production
fab -u ubuntu production setup_master -i aws-private.pem
```

This will install salt-master and update the master configuration file. The master will use a set of base states from https://github.com/caktus/margarita checked out at `/srv/margarita`.

As part of the master setup, a new GPG public/private key pair is generated. The private key remains on the master but the public version is exported and fetched back to the developer's machine. This will be put in `conf/<environment>.pub.gpg`. This will be used by all developers to encrypt secrets for the environment and needs to be committed into the repo.

3.3 Pillar Setup

Before your project can be deployed to a server, the code needs to be accessible in a git repository. Once that is done you should update `conf/pillar/<environment>.sls` to set the repo and branch for the environment. E.g., change this:

```
# FIXME: Update to the correct project repo
repo:
  url: git@github.com:CHANGEME/CHANGEME.git
  branch: master
```

to this:

```
repo:
  url: git@github.com:account/reponame.git
  branch: master
```

You also need to set `project_name` and `python_version` in `conf/pillar/project.sls`. The project template is set up for 3.4 by default. If you want to use 2.7, you will need to change `python_version` and make a few changes to requirements. In `requirements/production.txt`, change python3-memcached to python-memcached.

For the environment you want to setup you will need to set the `domain` in `conf/pillar/<environment>.sls`.

You will also need add the developer's user names and SSH keys to `conf/pillar/devs.sls`. Each user record (under the parent `users:` key) should match the format:

```
example-user:
  public_key:
   - ssh-rsa <Full SSH Public Key would go here>
```

Additional developers can be added later, but you will need to create at least one user for yourself.

3.4 Managing Secrets

Secret information such as passwords and API keys must be encrypted before being added to the pillar files. As previously noted, provisioning the master for the environment generates a public GPG key which is added to repo under `conf/<environment>.pub.gpg` To encrypt a new secret using this key, you can use the `encrypt` fab command:

```
# Example command
fab <environment> encrypt:<key>=<secret-value>
# Encrypt the SECRET_KEY for the staging environment
fab staging encrypt:SECRET_KEY='thisismysecretkey'
```

The output of this command will look something like:

```
"SECRET_KEY": |-
  -----BEGIN PGP MESSAGE-----
  Version: GnuPG v1.4.11 (GNU/Linux)
  hQEMA87BIemwflZuAQf/XDTq6pdZsS07zw88lvGcWbcy5pj5CLueVldE+NLAHilv
  YaFb1qPM1W+yrnxFQgsapcHUM82ULkXbMskYoK5qp5Or2ujwzAVRpbSrFTq19Frz
  sasFTPNNREgThLB8oyQIHN2XfqSvIqi6RkqXGf+eQDXLyl9Guu+7EhFtW5PJRo3i
  BSBVEuMi4Du60uAssQswNuit7lkEqxFprZDb9aHmjVBi+DAipmBuJ+FIyK0ePFAf
  dVfp/Es/y4/hWkM7TXDw5JMFtVfCo6Dm1LE53N339eJX01w19exB/Sek6HVwDsL4
  d45c1dm7qBiXN0zO8Yadhm520J0H9NcIPO47KyRkCtJAARsY5eu8cHxYW4DcYWLu
  PRr2CLuI8At1Q2KqlRgdEm17lV5HOEcMoT1SyvMzaWOnbpul5PoLCAebJ0zcJZT5
  Pw==
  =V1Uh
  -----END PGP MESSAGE-----
```

where `SECRET_KEY` would be replace with the key you were trying to encrypt. This block of text should be added to the environment pillar `conf/pillar/<environment>.sls` under the `secrets` block:

```
secrets:
  "SECRET_KEY": |-
    -----BEGIN PGP MESSAGE-----
    Version: GnuPG v1.4.11 (GNU/Linux)

    hQEMA87BIemwflZuAQf/XDTq6pdZsS07zw88lvGcWbcy5pj5CLueVldE+NLAHilv
    YaFb1qPM1W+yrnxFQgsapcHUM82ULkXbMskYoK5qp5Or2ujwzAVRpbSrFTq19Frz
    sasFTPNNREgThLB8oyQIHN2XfqSvIqi6RkqXGf+eQDXLyl9Guu+7EhFtW5PJRo3i
    BSBVEuMi4Du60uAssQswNuit7lkEqxFprZDb9aHmjVBi+DAipmBuJ+FIyK0ePFAf
    dVfp/Es/y4/hWkM7TXDw5JMFtVfCo6Dm1LE53N339eJX01w19exB/Sek6HVwDsL4
    d45c1dm7qBiXN0zO8Yadhm520J0H9NcIPO47KyRkCtJAARsY5eu8cHxYW4DcYWLu
    PRr2CLuI8At1Q2KqlRgdEm17lV5HOEcMoT1SyvMzaWOnbpul5PoLCAebJ0zcJZT5
    Pw==
    =V1Uh
    -----END PGP MESSAGE-----
```

The `Makefile` has a make command for generating a random secret. By default this is 32 characters long but can be adjusted using the `length` argument.:

```
make generate-secret
make generate-secret length=64
```

This can be combined with the above encryption command to generate a random secret and immediately encrypt it.:

```
fab staging encrypt:SECRET_KEY=`make generate-secret length=64`
```

By default the project will use the `SECRET_KEY` if it is set. You can also optionally set a `DB_PASSWORD`. If not set, you can only connect to the database server on localhost so this will only work for single server setups.

3.5 Github Deploy Keys

The repo will also need a deployment key generated so that the Salt minion can access the repository. You can generate a deployment key locally for the new server like so:

```
# Example command
make <environment>-deploy-key
# Generating the staging deploy key
make staging-deploy-key
```

This will generate two files named `<environment>.priv` and `conf/<environment>.pub.ssh`. The first file contains the private key and the second file contains the public key. The public key needs to be added to the "Deploy keys" in the GitHub repository. For more information, see the Github docs on managing deploy keys: https://help.github.com/articles/managing-deploy-keys

The text in the private key file should be added to *conf/pillar/<environment>.sls*‘ under the label *github_deploy_key* but it must be encrypted first. To encrypt the file you can use the same `encrypt` fab command as before passing the filename rather than a key/value pair:

```
fab staging encrypt:staging.priv
```

This will create a new file with appends `.asc` to the end of the original filename (i.e. staging.priv.asc). The entire contents of this file should be added to the `github_deploy_key` section of the pillar file.:

```
github_deploy_key: |
  -----BEGIN PGP MESSAGE-----
  Version: GnuPG v1.4.11 (GNU/Linux)

  hQEMA87BIemwflZuAQf/RW2bXuUpg5QuwuY9dLqLpdpKz+/971FHqM1Kz5NXgJHo
  hir8yh/wxlKlMbSpiyri6QPigj8DZLrGLi+VTwWCXJ
  ...
  -----END PGP MESSAGE-----
```

Do not commit the original `*.priv` files into the repo.

3.6 Environment Variables

Other environment variables which need to be configured but aren’t secret can be added to the `env` dictionary in `conf/pillar/<environment>.sls` without encryption.

# Additional public environment variables to set for the project env:

FOO: BAR

For instance the default layout expects the cache server to listen at `127.0.0.1:11211` but if there is a dedicated cache server this can be changed via `CACHE_HOST`. Similarly the `DB_HOST/DB_PORT` defaults to `''/''`:

```
env:
  DB_HOST: 10.10.20.2
  CACHE_HOST: 10.10.20.1:11211
```

3.7 Setup Checklist

To summarize the steps above, you can use the following checklist

- `repo` is set in `conf/pillar/<environment>.sls`
- Developer user names and SSH keys have been added to `conf/pillar/devs.sls`
- Project name has been set in `conf/pillar/project.sls`
- Environment domain name has been set in `conf/pillar/<environment>.sls`
- Environment secrets including the deploy key have been set in `conf/pillar/<environment>.sls`

3.8 Provision a Minion

Once you have completed the above steps, you are ready to provision a new server for a given environment. Again you will need to be able to connect to the server as a root user. This is to install the Salt Minion which will connect to the Master to complete the provisioning. To setup a minion you call the Fabric command:

```
fab <environment> setup_minion:<roles> -H <ip-of-new-server> -u <root-user>
fab staging setup_minion:web,balancer,db-master,cache -H  33.33.33.10 -u root
```

The available roles are `salt-master`, `web`, `worker`, `balancer`, `db-master`, `queue` and `cache`. If you are running everything on a single server you need to enable the `salt-master`, `web`, `balancer`, `db-master`, and `cache` roles. The `worker` and `queue` roles are only needed to run Celery which is explained in more detail later.

Additional roles can be added later to a server via `add_role`. Note that there is no corresponding `delete_role` command because deleting a role does not disable the services or remove the configuration files of the deleted role:

```
fab add_role:web -H  33.33.33.10
```

After that you can run the deploy/highstate to provision the new server:

```
fab <environment> deploy -u <root-user>
```

The first time you run this command, it may complete before the server is set up. It is most likely still completing in the background. If the server does not become accessible or if you encounter errors during the process, review the Salt logs for any hints in `/var/log/salt` on the minion and/or master. For more information about deployment, see the *server setup* *</server-setup>* documentation.

The initial deployment will create developer users for the server so you should not need to connect as root after the first deploy.

3.9 Optional Configuration

The default template contains setup to help manage common configuration needs which are not enabled by default.

3.9.1 HTTP Auth

The `<environment>.sls` can also contain a section to enable HTTP basic authentication. This is useful for staging environments where you want to limit who can see the site before it is ready. This will also prevent bots from crawling and indexing the pages. To enable basic auth simply add a section called `http_auth` in the relevant `conf/pillar/<environment>.sls`. As with other passwords this should be encrypted before it is added:

```
# Example encryption
fab <environment> encrypt:<username>=<password>
# Encrypt admin/abc123 for the staging environment
fab staging encrypt:admin=abc123
```

This would be added in `conf/pillar/<environment>.sls` under `http_auth`:

http_auth:

"admin": |- —–BEGIN PGP MESSAGE—– Version: GnuPG v1.4.11 (GNU/Linux)

hQEMA87BIemwflZuAQf+J4+G74ZSfrUPRF7z7+DPAmhBlK//A6dvplrsY2RsfEE4
Tfp7QPrHZc5V/gS3FXvlIGWzJOEFscKslzgzlccCHqsNUKE96qqnTNjsIoGOBZ4z
tmZV2F3AXzOVv4bOgipKIrjJDQcFJFjZKMAXa4spOAUp4cyIV/AQBu0Gwe9EUkfp
yXD+C/qTB0pCdAv5C4vyl+TJ5RE4fGnuPsOqzy4Q0mv+EkXf6EHL1HUywm3UhUaa
wbFdS7zUGrdU1BbJNuVAJTVnxAoM+AhNegLK9yAVDweWK6pApz3jN6YKfVLFWg1R
+miQe9hxGa2C/9X9+7gxeUagqPeOU3uX7pbUtJldwdJBAY++dkerVIihlbyWOkn4
0HYlzMI27ezJ9WcOV4ywTWwOE2+8dwMXE1bWlMCC9WAl8VkDDYup2FNzmYX87Kl4
9EY= =PrGi —–END PGP MESSAGE—–

This should be a list of key/value pairs. The keys will serve as the usernames and the values will be the password. As with all password usage please pick a strong password.

3.9.2 Celery

Many Django projects make use of Celery for handling long running tasks outside of the request/response cycle. Enabling a worker makes use of Django setup for Celery. As documented on that page, you need to create a new file in `school_navigator/celery.py` and then modify `school_navigator/__init__.py` to import that file. You'll also need to customize `{{ project_name}}/celery.py` to import the environment variables from `.env`. Add this (before the `os.environ.setdefault` call):

```
from . import load_env
load_env.load_env()
```

You should now be able to run the worker locally via (once you've added `celery` to your `requirements/base.txt` and installed it):

```
celery -A school_navigator worker
```

Additionally you will need to uncomment the `BROKER_URL` setting in the project settings:

```
# school_navigator/settings/deploy.py
from .base import *

# ...
BROKER_URL = 'amqp://school_navigator_%(ENVIRONMENT)s:%(BROKER_PASSWORD)s@%(BROKER_HOST)s/school_navi
```

These are the minimal settings to make Celery work. Refer to the Celery documentation for additional configuration options.

`BROKER_HOST` defaults to `127.0.0.1:5672`. If the queue server is configured on a separate host that will need to be reflected in the `BROKER_URL` setting. This is done by setting the `BROKER_HOST` environment variable in the `env` dictionary of `conf/pillar/<environment>.sls`.

To add the states you should add the `worker` role when provisioning the minion. At least one server in the stack should be provisioned with the `queue` role as well. This will use RabbitMQ as the broker by default. The RabbitMQ user will be named `school_navigator_<environment>` and the vhost will be named `school_navigator_<environment>` for each environment. It requires that you add a password for the RabbitMQ user to each of the `conf/pillar/<environment>.sls` under the secrets using the key `BROKER_PASSWORD`. As with all secrets this must be encrypted.

The worker will run also run the `beat` process which allows for running periodic tasks.

3.9.3 SSL

The default configuration expects the site to run under HTTPS everywhere. However, unless an SSL certificate is provided, the site will use a self-signed certificate. To include a certificate signed by a CA you must update the `ssl_key` and `ssl_cert` pillars in the environment secrets. The `ssl_cert` should contain the intermediate certificates provided by the CA. It is recommended that this pillar is only pushed to servers using the `balancer` role. See the `secrets.ex` file for an example.

You can use the below OpenSSL commands to generate the key and signing request:

```
# Generate a new 2048 bit RSA key
openssl genrsa -out school_navigator.key 2048
# Make copy of the key with the passphrase
cp school_navigator.key school_navigator.key.secure
# Remove any passphrase
openssl rsa -in school_navigator.secure -out school_navigator.key
# Generate signing request
openssl req -nodes -sha256 -new -key school_navigator.key -out school_navigator.csr
```

The last command will prompt you for information for the signing request including the organization for which the request is being made, the location (country, city, state), email, etc. The most important field in this request is the common name which must match the domain for which the certificate is going to be deployed (i.e example.com).

This signing request (.csr) will be handed off to a trusted Certificate Authority (CA) such as StartSSL, NameCheap, GoDaddy, etc. to purchase the signed certificate. The contents of the *.key file will be added to the `ssl_key` pillar and the signed certificate from the CA will be added to the `ssl_cert` pillar. These should be encrypted using the same proceedure as with the private SSH deploy key.

3.10 Quickstart

3.10.1 Production

```
fab -u ubuntu production setup_master -i ~/.ssh/aws-cfa.pem
rm production*.asc

fab production encrypt:DB_PASSWORD=`make generate-secret`
fab production encrypt:SECRET_KEY=`make generate-secret length=64`
fab production encrypt:BROKER_PASSWORD=`make generate-secret`
fab production encrypt:NEW_RELIC_LICENSE_KEY='<fill-me-in>'
fab production encrypt:production-ssl.key && cat production-ssl.key.asc
fab production encrypt:production-ssl.cert && cat production-ssl.cert.asc

fab production setup_minion:salt-master,web,balancer,db-master,cache,queue,worker -H ec2-52-2-56-101
fab production deploy -H ec2-52-2-56-101.compute-1.amazonaws.com -u ubuntu -i ~/.ssh/aws-cfa.pem
fab production deploy

# load db dump (run on server)
sudo supervisorctl stop all
sudo -u postgres dropdb school_navigator_production
sudo -u postgres createdb -E UTF-8 -O school_navigator_production school_navigator_production
sudo -u postgres psql -c 'CREATE EXTENSION postgis;' school_navigator_production
wget https://s3.amazonaws.com/school-navigator/db-2015-10-10.tar.zip
unzip db-2015-10-10.tar.zip
sudo -u postgres pg_restore -Ox -Ft --no-data-for-failed-tables -U school_navigator_production -d sch
sudo supervisorctl start all
```

CHAPTER 4

Vagrant Testing

4.1 Starting the VM

You can test the provisioning/deployment using Vagrant. This requires Vagrant 1.7+. The Vagrantfile is configured to install the Salt Master and Minion inside the VM once you've run `vagrant up`. The box will be installed if you don't have it already.:

```
vagrant up
```

The general provision workflow is the same as in the previous provisioning guide so here are notes of the Vagrant specifics.

4.2 Provisioning the VM

Set your environment variables and secrets in `conf/pillar/local.sls`. It is OK for this to be checked into version control because it can only be used on the developer's local machine. To finalize the provisioning you simply need to run:

```
fab vagrant setup_master
fab vagrant setup_minion:salt-master,db-master,cache,web,balancer -H 127.0.0.1:2222
fab vagrant deploy
```

The above command will setup Vagrant to run the full stack. If you want to test only a subset of the roles you can remove the unneeded roles. If you want to test the Celery setup then you can also add the `queue` and `worker` roles to the list.

The Vagrant box will use the current working copy of the project and the local.py settings. If you want to use this for development/testing it is helpful to change your local settings to extend from staging instead of dev:

```
# Example local.py
from school_navigator.settings.staging import *

# Override settings here
DATABASES['default']['NAME'] = 'school_navigator_local'
DATABASES['default']['USER'] = 'school_navigator_local'

DEBUG = True
```

This won't have the same nice features of the development server such as auto-reloading but it will run with a stack which is much closer to the production environment. Also beware that while `conf/pillar/local.sls` is checked into version control, `local.py` generally isn't, so it will be up to you to keep them in sync.

4.3 Testing on the VM

With the VM fully provisioned and deployed, you can access the VM at the IP address specified in the `Vagrantfile`, which is 33.33.33.10 by default. Since the Nginx configuration will only listen for the domain name in `conf/pillar/local.sls`, you will need to modify your `/etc/hosts` configuration to view it at one of those IP addresses. I recommend 33.33.33.10, otherwise the ports in the localhost URL cause the CSRF middleware to complain `REASON_BAD_REFERER` when testing over SSL. You will need to add:

```
33.33.33.10 <domain>
```

where `<domain>` matches the domain in `conf/pillar/local.sls`. For example, let's use dev.example.com:

```
33.33.33.10 dev.example.com
```

In your browser you can now view https://dev.example.com and see the VM running the full web stack.

Note that this `/etc/hosts` entry will prevent you from accessing the true dev.example.com. When your testing is complete, you should remove or comment out this entry.

CHAPTER 5

Indices and tables

- genindex
- modindex
- search