NC Traffic Stops Documentation

Release 0.1

Dylan, Andy, and Colin

Nov 13, 2018

Contents

Open Data Policing is a project of the Southern Coalition for Social Justice. The site's development team consists of attorney Ian Mance of the Southern Coalition; Colin Copeland, CTO of Caktus Group, and volunteer developers Andy Shapiro and Dylan Young of Durham, NC. A special thanks to Tom Meehan for assisting with analyzing US census data.

This is the developer documentation.

Contents:

CHAPTER 1

Development Setup

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

- Python 3.4
- NodeJS >= 4.2
- pip >= 8 or so
- virtualenv >= 1.10
- virtualenvwrapper >= 3.0
- Postgres >= 9.3
- git >= 1.7

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
```

(If you build Python 3.4 yourself on Ubuntu, ensure that the *libbz2-dev* package is installed first.)

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==1.10.0
```

For a working `fab encrypt` you'll need more modules in a Python 2 environment. Create a new virtualenv for that and use `requirements/fab.txt`.

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.

1.1 Getting Started

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

```
$ which python3.4  # make sure you have Python 3.4 installed
$ mkvirtualenv --python=`which python3.4` opendatapolicing
(opendatapolicing)$ pip install -U pip
(opendatapolicing)$ pip install -r requirements/dev.txt
(opendatapolicing)$ npm install
```

If `npm install` fails, make sure you're using `npm` from a reasonable version of NodeJS, as documented at the top of this document.

Next, we'll set up our local environment variables. We use django-dotenv to help with this. It reads environment variables located in a file name `.env` in the top level directory of the project. The only variable we need to start is `DJANGO_SETTINGS_MODULE`:

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

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

```
(opendatapolicing)$ deactivate
(opendatapolicing)$ workon opendatapolicing
```

Create the Postgres database and run the initial syncdb/migrate:

```
(opendatapolicing)$ createdb -E UTF-8 traffic_stops
(opendatapolicing)$ createdb -E UTF-8 traffic_stops_nc
(opendatapolicing)$ createdb -E UTF-8 traffic_stops_md
(opendatapolicing)$ createdb -E UTF-8 traffic_stops_il
(opendatapolicing)$ ./migrate_all_dbs.sh
```

1.2 Development

You should be able to run the development server via the configured `dev` script:

```
(opendatapolicing)$ npm run dev
```

Or, on a custom port and address:

```
(opendatapolicing)$ npm run dev -- --address=0.0.0.0 --port=8020
```

Any changes made to Python, Javascript or Less files will be detected and rebuilt transparently as long as the development server is running.

1.3 When running migrations

This is a multi-database project. Whenever you have unapplied migrations, either added locally or via an update from the source repository, the migrations need to be applied to all databases by running the `./migrate_all_dbs.sh` command.

CHAPTER 2

Docker

You can use the provided `docker-compose` environment to create a local development environment. See previous section for more detailed instructions about how this project is configured.

Setup your local development settings:

```
cp traffic_stops/settings/local.example.py traffic_stops/settings/local.py
# uncomment lines below "UNCOMMENT BELOW IF USING DOCKER SETUP" in local.py
echo "DJANGO_SETTINGS_MODULE=traffic_stops.settings.local" > .env
```

For basic setup, run the following commands:

```
docker-compose up -d db  # start the PostgreSQL container in the background
docker-compose build web  # build the container (can take a while)
docker-compose run --rm web createdb -E UTF-8 traffic_stops
docker-compose run --rm web createdb -E UTF-8 traffic_stops_nc
docker-compose run --rm web createdb -E UTF-8 traffic_stops_md
docker-compose run --rm web createdb -E UTF-8 traffic_stops_il
docker-compose run --rm web ./migrate_all_dbs.sh
```

Run `npm install` inside the `web` container (you'll need to do this for any update to `package.json`):

```
rm -rf ./node_modules  # if needed
docker-compose run --rm web bash -lc "npm install"
```

You can now run the web container and tail the logs:

```
# start up the dev server, and watch the logs:
docker-compose up -d web && docker-compose logs -f web
```

These are other useful docker-compose commands:

```
# explicitly execute runserver in the foreground (for breakpoints):
docker-compose stop web
docker-compose run --rm --service-ports web python manage.py runserver 0.0.0.0:8000
```

Visit http://localhost:8003/ in your browser.

2.1 Restore Production Data

The data import process for each state can take a long time. You can load the production data using the following steps:

First download a dump (in this case, NC) of the database:

```
ssh opendatapolicing.com 'sudo -u postgres pg_dump -Fc -Ox traffic_stops_nc_production
↪' > traffic_stops_nc_production.pgdump
```

Now run `pg_restore` within the `web` container:

```
docker-compose stop web  # free up connections to the DB
docker-compose run --rm web dropdb traffic_stops_nc
docker-compose run --rm web createdb -E UTF-8 traffic_stops_nc
docker-compose run --rm web pg_restore -Ox -d traffic_stops_nc traffic_stops_nc_
↪production.pgdump
rm traffic_stops_nc_production.pgdump  # so it doesn't get built into the container
```

You can also load the primary DB with user accounts and state statistics:

```
ssh opendatapolicing.com 'sudo -u postgres pg_dump -Fc -Ox traffic_stops_production' >
↪ traffic_stops_production.pgdump
docker-compose stop web  # free up connections to the DB
docker-compose run --rm web dropdb traffic_stops
docker-compose run --rm web createdb -E UTF-8 traffic_stops
docker-compose run --rm web pg_restore -Ox -d traffic_stops traffic_stops_production.
↪pgdump
rm traffic_stops_production.pgdump  # so it doesn't get built into the container
```

2.2 Deployment

You can run a deployment from within a docker container using the following commands:

```
docker-compose run --rm web /bin/bash
eval $(ssh-agent)
ssh-add ~/.ssh/YOUR_KEY

fab -u YOUR_USER staging salt:"test.ping"
```

CHAPTER 3

Data Import

Stop data can be imported in the same manner for all states. Substitute the state abbreviation (e.g., "md") as appropriate in the Generic NC instructions below.

Census data for all states is imported all at once, in the same manner for all environments, using the `import_census` management command. This must be performed as part of developer and server setup as well as when census support is added for additional states.

3.1 Local/Development Environment

Before running state imports, first import census data:

```
python manage.py import_census
```

3.1.1 Database Dump (quicker)

To load an existing database dump on S3, run:

```
dropdb traffic_stops_nc
createdb -E UTF-8 traffic_stops_nc
wget https://s3-us-west-2.amazonaws.com/openpolicingdata/traffic_stops_nc_2018_01_08.
↪dump.zip
unzip traffic_stops_nc_2018_01_08.dump.zip
pg_restore -Ox -d traffic_stops_nc traffic_stops_nc_2018_01_08.dump
```

Browse https://s3-us-west-2.amazonaws.com/openpolicingdata/ to see what dumps are available.

To create a new database dump, run:

```
ssh dev.opendatapolicingnc.com 'sudo -u postgres pg_dump -Fc traffic_stops_nc_staging
↪' > traffic_stops_nc.dump
```

That can be loaded with the `pg_restore` command shown above.

3.1.2 Raw NC Data (slower)

The state-specific database must exist and current migrations need to have been applied before importing. If in doubt:

```
# for NC
dropdb traffic_stops_nc && createdb -E UTF-8 traffic_stops_nc
# for MD
dropdb traffic_stops_md && createdb -E UTF-8 traffic_stops_md

./migrate_all_dbs.sh
```

Command-line

If loading NC, make sure to add `NC_FTP_USER` and `NC_FTP_PASSWORD` and your `.env` file.

If on a Mac, install `gnu-sed`:

```
brew install gnu-sed --with-default-names
```

Run the import command:

```
# for NC (~25m)
rm -rf ./ncdata  # if you don't want to reuse previous download
python manage.py import_nc --dest $PWD/ncdata --noprime  # noprime = don't prime cache
# for MD (~30m)
rm -rf ./mddata  # if you don't want to reuse previous download
python manage.py import_md --dest $PWD/mddata
```

This took ~25 minutes on my laptop. Run `tail -f traffic_stops.log` to follow along. Reusing an existing `--dest` directory will speed up import. However, if import code has changed since the last time the directory was used, don't reuse an existing directory.

Now you should be able to view data with `runserver`:

```
python manage.py runserver
```

Admin

Access /admin/tsdata/dataset/ and create a "dataset" describing the data to be imported. Setting the fields:

- Select the desired state
- Provide a unique name for the dataset
- The date received should reflect when the raw data was received
- Set the URL to one of the available datasets at https://s3-us-west-2.amazonaws.com/openpolicingdata/ . The normal URLs are stored in the source code (in `<state_app>.data.__init__.py`). For NC, if you use the magic URL `ftp://nc.us/`, the latest available dataset will be downloaded from the state and used for this import.
- Specify a destination directory where the dataset will be downloaded and extracted.
- Optionally specify one or two e-mail addresses that will be notified when the import completes successfully.

Once the "dataset" has been created, select the new dataset in list view and apply the "Import selected dataset" action.

3.2 Server

The PostgreSQL user must have SUPERUSER privileges to perform the import. Depending on current admin policies, that may have to be granted and revoked around the import.

Temporarily grant our PostgreSQL user SUPERUSER privileges:

```
sudo -u postgres psql -c 'ALTER USER traffic_stops_staging WITH SUPERUSER;'
```

When finished, revoke SUPERUSER privileges:

```
sudo -u postgres psql -c 'ALTER USER traffic_stops_staging WITH NOSUPERUSER;'
```

When importing IL data on a server, paging space is required due to the memory requirements. Currently the staging and production servers do not have a "swap" file or device permanently assigned, nor do they have a device on which paging space can be routinely used without incurring I/O charges. Thus a swap file is activated prior to an import of IL data and then deactivated afterwards, as follows:

```
sudo fallocate -l 3G /swapfile
sudo chmod 600 /swapfile
sudo mkswap /swapfile
sudo swapon /swapfile
<<perform the IL data import using the appropriate mechanism>>
sudo swapoff /swapfile
sudo rm /swapfile
```

3.2.1 Raw NC Data

Command-line

Run the import command:

```
sudo su - traffic_stops
cd /var/www/traffic_stops
source ./env/bin/activate
./manage.sh import_nc --dest=/var/www/traffic_stops/data
```

Reusing an existing `--dest` directory will speed up import. However, if import code has changed since the last time the directory was used, don't reuse an existing directory.

Admin

Follow the "Admin" instructions above under "Local/Development Environment".

3.2.2 Create DB Dump

```
sudo -u postgres pg_dump -Ox -Ft traffic_stops_nc_production > traffic_stops_nc_
↪production.tar
zip traffic_stops_nc_production.tar.zip traffic_stops_nc_production.tar
# then on local laptop, run:
scp opendatapolicingnc.com:traffic_stops_nc_production.tar.zip .
```

3.3 Updating landing page stats

NC landing page stats are updated automatically after import. This section applies only to other states. (The NC command in the example below will work and can be used during development, but for NC it is not necessary to run the command and update the Django template using the output when you get a new set of data from the state.)

Currently, various statistics on the state landing page are hard-coded in the Django templates for that state, including the number of stops, the range of dates, and the top five agencies.

When first importing a new set of data from a state, the landing page stats must be edited to reflect the new data. This process involves the following steps:

1. Calculate the statistics using the new dataset.
2. Update the Django template for the state to include the current statistics.
3. Pay attention to whether or not agency ids or the top five agencies have changed; if they have, the top five agencies as shown in the landing page will require more editing.

The landing page stats are computed with the `<state_app>_dataset_facts` management commands. Example:

```
$ ./manage.py nc_dataset_facts
Timeframe: Jan 01, 2000 - Apr 12, 2016
Stops: 20,622,253
Searches: 632,719
Agencies: 314

Top 5:
Id 193: NC State Highway Patrol 9,608,578
Id 51: Charlotte-Mecklenburg Police Department 1,600,836
Id 224: Raleigh Police Department 863,653
Id 104: Greensboro Police Department 555,453
Id 88: Fayetteville Police Department 503,013
```

CHAPTER 4

Server Setup

4.1 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 doc:*provisioning guide </provisioning>*.

4.2 Layout

Below is the server layout created by this provisioning process:

```
/var/www/traffic_stops/
    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.

4.3 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.

CHAPTER 5

Server Provisioning

5.1 Overview

traffic_stops is deployed on the following stack.

- OS: Ubuntu 14.04 LTS
- Python: 3.4
- Database: Postgres 9.3, PostGIS 2.1
- 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.

5.2 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 33.33.33.10 as the Salt Master for staging
fab -u root staging setup_master
# Example AWS setup
fab -u ubuntu -i ~/.ssh/traffic-stops.pem staging setup_master
```

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.

5.3 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:copelco/NC-Traffic-Stops.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.

5.4 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.

5.5 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.

5.6 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
```

5.7 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`

5.8 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
# Example AWS setup
fab staging setup_minion:web,balancer,db-master,cache,queue,worker -H 52.6.26.10 -u␣
↪ubuntu -i ~/.ssh/traffic-stops.pem
fab staging deploy -H 52.6.26.10 -u ubuntu -i ~/.ssh/traffic-stops.pem
```

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 `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
```

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.

5.9 Optional Configuration

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

5.9.1 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.

5.9.2 Celery

Many Django projects make use of Celery for handling long running task outside of request/response cycle. Enabling a worker makes use of Django setup for Celery. As documented you should create/import your Celery app in `traffic_stops/__init__.py` so that you can run the worker via:

```
celery -A traffic_stops worker
```

Additionally you will need to configure the project settings for Celery:

```
# traffic_stops.settings.staging.py
import os
from traffic_stops.settings.base import *

# Other settings would be here
BROKER_URL = 'amqp://traffic_stops_staging:%(BROKER_PASSWORD)s@%(BROKER_HOST)s/
↪traffic_stops_staging' % os.environ
```

You will also need to add the `BROKER_URL` to the `traffic_stops.settings.production` so that the vhost is set correctly. 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 traffic_stops_<environment> and the vhost will be named traffic_stops_<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.

5.9.3 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 traffic_stops.key 2048
# Make copy of the key with the passphrase
cp traffic_stops.key traffic_stops.key.secure
# Remove any passphrase
openssl rsa -in traffic_stops.secure -out traffic_stops.key
# Generate signing request
openssl req -new -key traffic_stops.key -out traffic_stops.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.

5.10 Quickstart

5.10.1 Staging

Terraform:

```
terraform plan -var-file="secrets.tfvars" -var-file="staging.tfvars"
terraform apply -var-file="secrets.tfvars" -var-file="staging.tfvars"

terraform plan -destroy -var-file="secrets.tfvars" -var-file="staging.tfvars"
terraform destroy -var-file="secrets.tfvars" -var-file="staging.tfvars"
```

Salt:

```
ssh-keygen -f "$HOME/.ssh/known_hosts" -R dev.opendatapolicingnc.com
ssh-keygen -f "$HOME/.ssh/known_hosts" -R 52.6.26.10
fab -u ubuntu -i ~/.ssh/traffic-stops.pem staging setup_master
fab staging encrypt:DB_PASSWORD=`pwgen --secure -1 32`
fab staging encrypt:SECRET_KEY=`pwgen --secure -1 64`
fab staging encrypt:BROKER_PASSWORD=`pwgen --secure -1 32`
fab staging encrypt:LOG_DESTINATION='<fill-me-in>'
fab staging encrypt:admin='<fill-me-in>'
fab staging encrypt:NEW_RELIC_LICENSE_KEY='<fill-me-in>'
# copy each generated encrypted key to conf/pillar/<env>.sls
fab staging setup_minion:web,balancer,db-master,cache,queue,worker,salt-master -H dev.
↪opendatapolicingnc.com -u ubuntu -i ~/.ssh/traffic-stops.pem
fab staging deploy -H dev.opendatapolicingnc.com -u ubuntu -i ~/.ssh/traffic-stops.pem
fab staging deploy
```

5.10.2 Production

Terraform:

```
make production  # see plan
make production-apply
```

Salt:

```
ssh-keygen -f "$HOME/.ssh/known_hosts" -R opendatapolicing.com
ssh-keygen -f "$HOME/.ssh/known_hosts" -R 52.206.92.217
fab -u ubuntu -i ~/.ssh/traffic-stops.pem production setup_master
rm production*.asc
fab production encrypt:DB_PASSWORD=`pwgen --secure -1 32`
fab production encrypt:SECRET_KEY=`pwgen --secure -1 64`
fab production encrypt:BROKER_PASSWORD=`pwgen --secure -1 32`
fab production encrypt:production-ssl.cert && cat production-ssl.cert.asc
fab production encrypt:production-ssl.key && cat production-ssl.key.asc
fab production encrypt:admin=<fill-me-in>
fab production encrypt:LOG_DESTINATION='<fill-me-in>'
fab production encrypt:NEW_RELIC_LICENSE_KEY='<fill-me-in>'
# copy each generated encrypted key to conf/pillar/<env>.sls
fab production setup_minion:web,balancer,db-master,cache,queue,worker,salt-master -H␣
↪opendatapolicing.com -u ubuntu -i ~/.ssh/traffic-stops.pem
fab production deploy -H opendatapolicing.com -u ubuntu -i ~/.ssh/traffic-stops.pem
fab production deploy
```

CHAPTER 6

API Endpoints

6.1 Stops by all races and ethnicities by year

URI: /api/agency/<id>/stops/

Officer URI: /api/agency/<id>/stops/?officer=<id>

Counts of stops by all races and by all ethnicities by year.

6.1.1 SQL

Sample SQL query (Durham Police Department):

```
SELECT count(person_id),
       p.race,
       extract(YEAR FROM s.date) AS year
FROM stops_person p
JOIN stops_stop s ON p.stop_id = s.stop_id
WHERE p.type='D'
  AND s.agency_id = 78
GROUP BY p.race,
         year
ORDER BY year ASC, p.race DESC;
```

Sample SQL Results:

```
 count | race | year
-------+------+------
  4481 | W    | 2005
   357 | U    | 2005
     9 | I    | 2005
  5665 | B    | 2005
   163 | A    | 2005
```

(continues on next page)

(continued from previous page)

```
  5319 | W    | 2006
   231 | U    | 2006
    41 | I    | 2006
  7205 | B    | 2006
   178 | A    | 2006
  7520 | W    | 2007
   120 | U    | 2007
    75 | I    | 2007
 10372 | B    | 2007
   261 | A    | 2007
```

6.1.2 JSON

Sample JSON response (Durham Police Department):


```
[
   {
        "year": 2005,
        "native_american": 9,
        "black": 5665,
        "white": 4481,
        "other": 357,
        "non-hispanic": 9298,
        "hispanic": 1377,
        "asian": 163
    },
    {
        "year": 2006,
        "native_american": 41,
        "black": 7200,
        "white": 5318,
        "other": 231,
        "non-hispanic": 11342,
        "hispanic": 1626,
        "asian": 178
    },
    {
        "year": 2007,
        "native_american": 75,
        "black": 10365,
        "white": 7516,
        "other": 120,
        "non-hispanic": 16050,
        "hispanic": 2287,
        "asian": 261
    },
]
```


6.2 Likelihood-of-search by stop-reason

URI: /api/agency/<id>/stops_by_reason/

Officer URI: /api/agency/<id>/stops_by_reason/?officer=<id>

A count of likelihood-of-search by stop-reason.

6.2.1 SQL Query

One query for all stops and another for only stops with searches.

```
SELECT count(p.person_id),
       p.race,
       s.purpose,
       extract(YEAR FROM s.date) AS year
FROM stops_person p
JOIN stops_stop s ON p.stop_id = s.stop_id
WHERE p.type='D'
  AND s.agency_id = 78
GROUP BY p.race,
         s.purpose,
         year
ORDER BY year ASC,
         s.purpose ASC,
         p.race DESC;

SELECT count(se.person_id),
       p.race,
       s.purpose,
       extract(YEAR FROM s.date) AS year
FROM stops_person p
JOIN stops_stop s ON p.stop_id = s.stop_id
JOIN stops_search se ON s.stop_id = se.stop_id
WHERE p.type='D'
  AND s.agency_id = 78
GROUP BY p.race,
         s.purpose,
         year
ORDER BY year ASC,
         s.purpose ASC,
         p.race DESC;
```

Sample SQL Results:

```
 count | race | purpose | year
-------+------+---------+------
  2568 | W    |       1 | 2006
   134 | U    |       1 | 2006
    31 | I    |       1 | 2006
  2386 | B    |       1 | 2006
   117 | A    |       1 | 2006
   272 | W    |       2 | 2006
    18 | U    |       2 | 2006
   348 | B    |       2 | 2006
     8 | A    |       2 | 2006
    29 | W    |       3 | 2006
    35 | B    |       3 | 2006
   342 | W    |       4 | 2006
     9 | U    |       4 | 2006
     1 | I    |       4 | 2006
   430 | B    |       4 | 2006
    11 | A    |       4 | 2006
```

(continues on next page)

(continued from previous page)

```
   628 | W    |        5 | 2006
    14 | U    |        5 | 2006
     3 | I    |        5 | 2006
  1231 | B    |        5 | 2006
    12 | A    |        5 | 2006
   750 | W    |        6 | 2006
    20 | U    |        6 | 2006
     4 | I    |        6 | 2006
  1511 | B    |        6 | 2006
    11 | A    |        6 | 2006
   198 | W    |        7 | 2006
     9 | U    |        7 | 2006
   373 | B    |        7 | 2006
     5 | A    |        7 | 2006
   204 | W    |        8 | 2006
     3 | U    |        8 | 2006
   409 | B    |        8 | 2006
     1 | A    |        8 | 2006
   328 | W    |        9 | 2006
    24 | U    |        9 | 2006
     2 | I    |        9 | 2006
   482 | B    |        9 | 2006
    13 | A    |        9 | 2006

 count | race | purpose | year
-------+------+---------+------
    73 | W    |        1 | 2006
     1 | U    |        1 | 2006
   126 | B    |        1 | 2006
     5 | A    |        1 | 2006
    21 | W    |        2 | 2006
     1 | U    |        2 | 2006
    25 | B    |        2 | 2006
    19 | W    |        3 | 2006
    18 | B    |        3 | 2006
    44 | W    |        4 | 2006
    56 | B    |        4 | 2006
    62 | W    |        5 | 2006
   156 | B    |        5 | 2006
     1 | A    |        5 | 2006
    47 | W    |        6 | 2006
     1 | U    |        6 | 2006
   169 | B    |        6 | 2006
     5 | W    |        7 | 2006
     1 | U    |        7 | 2006
    26 | B    |        7 | 2006
    29 | W    |        8 | 2006
    91 | B    |        8 | 2006
     1 | A    |        8 | 2006
    16 | W    |        9 | 2006
     2 | U    |        9 | 2006
     1 | I    |        9 | 2006
    50 | B    |        9 | 2006
```

6.2.2 JSON Response


```
{
    "searches": [
        {
            "purpose": "Speed Limit Violation",
            "year": 2006,
            "hispanic": 35,
            "native_american": 0,
            "white": 73,
            "asian": 5,
            "black": 126,
            "non-hispanic": 170,
            "other": 1
        },
        {
            "purpose": "Stop Light/Sign Violation",
            "year": 2006,
            "hispanic": 14,
            "native_american": 0,
            "white": 21,
            "asian": 0,
            "black": 25,
            "non-hispanic": 33,
            "other": 1
        }
    ],
    "stops": [
        {
            "purpose": "Speed Limit Violation",
            "year": 2006,
            "hispanic": 475,
            "native_american": 31,
            "white": 2567,
            "asian": 117,
            "black": 2386,
            "non-hispanic": 4760,
            "other": 134
        },
        {
            "purpose": "Stop Light/Sign Violation",
            "year": 2006,
            "hispanic": 90,
            "native_american": 0,
            "white": 272,
            "asian": 8,
            "black": 348,
            "non-hispanic": 556,
            "other": 18
        },
    ]
}
```


6.3 Use-of-force

URI: /api/agency/<id>/use_of_force/

Officer URI: /api/agency/<id>/use_of_force/?officer=<id>

A count of all use-of-force by all races and by all ethnicities by year.

6.3.1 SQL Query

Sample SQL query:

```
SELECT count(se.person_id),
       p.race,
       extract(YEAR FROM s.date) AS year
FROM stops_person p
JOIN stops_stop s ON p.stop_id = s.stop_id
JOIN stops_search se ON s.stop_id = se.stop_id
WHERE p.type='D'
  AND s.agency_id = 78
  AND s.engage_force = 't'
GROUP BY p.race,
         year
ORDER BY p.race DESC,
         year ASC;
```

Sample SQL results:

```
 count | race | year
-------+------+------
     3 | W    | 2002
     1 | W    | 2003
     1 | W    | 2005
     3 | W    | 2006
     3 | W    | 2007
     9 | W    | 2008
     1 | W    | 2010
     1 | W    | 2011
     1 | W    | 2012
     2 | U    | 2002
    12 | B    | 2002
     4 | B    | 2003
     4 | B    | 2004
     1 | B    | 2005
     5 | B    | 2006
    10 | B    | 2007
    12 | B    | 2008
     3 | B    | 2009
     4 | B    | 2010
     8 | B    | 2011
     4 | B    | 2012
     1 | B    | 2013
(22 rows)
```

6.3.2 JSON

Sample JSON response (Durham Police Department):

```
[
    {
        "year": 2006,
        "native_american": 0,
        "other": 0,
        "black": 5,
        "hispanic": 3,
        "asian": 0,
        "non-hispanic": 5,
        "white": 3
    },
    {
        "year": 2007,
        "native_american": 0,
        "other": 0,
        "black": 10,
        "hispanic": 1,
        "asian": 0,
        "non-hispanic": 12,
        "white": 3
    },
    {
        "year": 2008,
        "native_american": 0,
        "other": 0,
        "black": 12,
        "hispanic": 6,
        "asian": 0,
        "non-hispanic": 15,
        "white": 9
    }
]
```


6.4 Contraband Hit Rate

URI: /api/agency/<id>/contraband_hit_rate/

Officer URI: /api/agency/<id>/contraband_hit_rate/?officer=<id>

A count of contraband hit-rate by year and race.

6.4.1 SQL Query

One query for all stops with searches and another for stops with searches with contraband.

```
SELECT count(se.person_id),
       p.race,
       extract(YEAR FROM s.date) AS year
FROM stops_person p
JOIN stops_stop s ON p.stop_id = s.stop_id
JOIN stops_search se ON s.stop_id = se.stop_id
WHERE p.type='D'
  AND s.agency_id = 78
GROUP BY p.race,
```

(continues on next page)

(continued from previous page)

```
         year
ORDER BY year ASC,
         p.race DESC;

SELECT count(c.person_id),
       p.race,
       extract(YEAR FROM s.date) AS year
FROM stops_person p
JOIN stops_stop s ON p.stop_id = s.stop_id
JOIN stops_search se ON s.stop_id = se.stop_id
JOIN stops_contraband c ON se.search_id = c.search_id
WHERE p.type='D'
  AND s.agency_id = 78
GROUP BY p.race,
         year
ORDER BY year ASC,
         p.race DESC;
```

Sample SQL Results:

```
 count | race | year
-------+------+------
   316 | W    | 2006
     6 | U    | 2006
     1 | I    | 2006
   717 | B    | 2006
     7 | A    | 2006
   465 | W    | 2007
     5 | U    | 2007
     3 | I    | 2007
   934 | B    | 2007
    17 | A    | 2007

 count | race | year
-------+------+------
    47 | W    | 2006
     1 | U    | 2006
   150 | B    | 2006
     2 | A    | 2006
    85 | W    | 2007
     1 | I    | 2007
   259 | B    | 2007
     4 | A    | 2007
```

6.4.2 JSON

Sample JSON response (Durham Police Department):

```
{
    "contraband": [
        {
            "year": 2006,
            "hispanic": 17,
            "native_american": 0,
            "other": 1,
```

(continues on next page)

(continued from previous page)


```
            "black": 149,
            "asian": 2,
            "non-hispanic": 182,
            "white": 47
        },
        {
            "year": 2007,
            "hispanic": 31,
            "native_american": 1,
            "other": 0,
            "black": 260,
            "asian": 4,
            "non-hispanic": 319,
            "white": 85
        },
    ],
    "searches": [
        {
            "year": 2006,
            "hispanic": 174,
            "native_american": 1,
            "other": 6,
            "black": 716,
            "asian": 7,
            "non-hispanic": 872,
            "white": 316
        },
        {
            "year": 2007,
            "hispanic": 225,
            "native_american": 3,
            "other": 5,
            "black": 935,
            "asian": 17,
            "non-hispanic": 1200,
            "white": 465
        },
    ]
}
```

CHAPTER 7

Vagrant Testing

7.1 Starting the VM

You can test the provisioning/deployment using Vagrant. This requires Vagrant 1.3+. 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.

7.2 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 deploy
```

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 traffic_stops.settings.staging import *

# Override settings here
DATABASES['default']['NAME'] = 'traffic_stops_local'
DATABASES['default']['USER'] = 'traffic_stops_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.

7.3 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/staging/env.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/staging/env.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 8

Indices and tables

- genindex
- modindex
- search