TracPro Documentation

Release 1.4.0

UNICEF

March 28, 2016

TracPro is a minimal, generic dashboard for polls in any RapidPro org, such as for a real-time monitoring service for education. TracPro is for simple dashboards that help to create information loops, not for advocacy or flashy visuals. TracPro is open source, anyone can build their own dashboard. Deployment requires advanced Linux systems administration experience, but once running it can be configured and used by anyone.

TracPro was built for UNICEF by Nyaruka and is currently maintained by Caktus Group.

CHAPTER 1

Technical Overview

1.1 Concepts

- *Polls* are RapidPro flows. Click on the name of a poll to see a summary of the responses over time and dates it was conducted.
- *PollRuns* are the dates a poll was conducted. Click on a poll date to see a summary of just the responses for that date, participation among reporter groups, send a message to contacts, individual responses, and export responses.
- *Contacts* are the same as in RapidPro, they are the respondents of polls.
- *Supervisors* could be, for example, a provincial monitoring officer in a ministry. They are meant to see a particular region or multiple regions, and encourage participation by seeing who is responding, and restart polls. Supervisors also manage contacts.
- *Administrators* setup TracPro for an org and can manage everything supervisors can, in addition to managing supervisors.
- *Regions* are administrative areas, and can be anything, for example a city, county, or state in the USA. They must be configured as contact groups in RapidPro first. Regions can be made hierarchical in TracPro. For example, a supervisor who has access only to Kampala, would not have access to all of Africa's regional data, and a second supervisor who had access to Africa, would have access to data for all sub-regions within Africa.
- *Reporter groups* are types of contacts, such as head teachers, or grade 5 teachers. They must be configured as contact groups in RapidPro first.
- *Recent Indicators* allow users to compare baseline poll results to follow up poll results over time. For instance, if a poll was sent out to gather enrollment numbers at the start of the term, that could be selected as the baseline poll, and a follow up poll could be the poll sent out over the term asking for attendance numbers. Users may also spoof poll data to create sample results to chart on Recent Indicators.
- *Inbox Messages* allow users to view and send unsolicited messages, that is, messages outside of a flow, to contacts.

1.2 Features

Administrator accounts: Configure a dashboard for any org, such as choosing polls, reporter groups, and regions to make available.

Supervisor management: Administrators can easily add, remove, and configure supervisors. Administators set up any desired access to particular regions for supervisors. Supervisors have access to data for their regions and to Homepage Poll data, Contacts, Inbox Messages and Recent Indicators.

Contact management: Supervisors can create, manage, and remove contacts, including changing their region, reporter group, facility and language, and all those changes are synchronized in the background with RapidPro.

Start polls: Supervisors can start any poll brought into TracPro from the Homepage.

Send messages: Send messages about polls to non-responsive contacts, responsive contacts or all contacts of a group. Select a Poll, select a date that the poll ran from the list of dates it was conducted on (PollRuns), choose the Participation tab, and click the button to "Send Message..." and choose whether to send it to respondents or non-respondents. These messages can be viewed from the Message Log.

Activity: See the most active regions and the most active reporter groups.

Participation: See participation of every contact in all polls on one page.

Right-to-left (RTL) language support: Any language can be displayed, whether left-to-right or right-to-left with content switching correctly too.

Easy translations: TracPro uses Transifex.com, the easy way to add a translation. Anyone can add translations that are then available for others to use in their TracPro dashboard.

Automatic charts: Simple analytics are automatically created because TracPro and RapidPro know what the type of response is for a flow step, i.e. numeric, multiple choice, open-ended text, or audio recording from IVR. Responses can be seen for time-series or for one poll issue.

Export as CSV: Export data from TracPro into CSV format, which can easily be imported into Microsoft Excel or other tools.

Edit poll names: Change any poll description.

Mobile-ready: TracPro works in smartphone and tablet browsers.

CHAPTER 2

Getting Started

The following steps take you through the process of creating a new organization in TracPro. If you are using your local development installation, then ensure that both the web server and Celery are running.

2.1 Preparing RapidPro

Before setting up your TracPro organization, one should ensure that the RapidPro organization has the following:

- A set of contact groups representing geographical regions, e.g. *Kigali*, *Florida*
- A set of contact groups representing reporting groups, e.g. *Males*, *Teachers*

Obviously you will also want to define some flows in RapidPro which are suitable for running as polls.

2.2 Creating An Organization In TracPro

- Navigate to http://localhost:8000/
- Log in as a superuser
- Navigate to http://localhost/manage/user/ and add a new administrator user account
- Navigate to http://localhost/manage/org/ and click *Add* to create a new organization
- Include the newly created user as an administrator
- Use the API token provided by your RapidPro organization. If you don't know it then visit the API explorer.
- Save new org and navigate to http://SUBDOMAIN.localhost:8000/ where SUBDOMAIN is the subdomain of your new organization
- Log in as the new administrator user

2.3 Configuring An Organization

There won't be much to see until you tell TracPro about which flows and groups to use.

- Navigate to **Administration > Polls** and click **Select** to select which flows in RapidPro will be used as polls in TracPro

- Navigate to **Administration > Reporter Groups** and click **Select** to select which contact groups in RapidPro will be used as reporter groups in TracPro
- Navigate to **Administration > Regions** and click **Select** to select which contact groups in RapidPro will be used as regions in TracPro. This will trigger a fetch of all contacts from those groups.

2.4 Fetching old runs

If a new poll is added, TracPro will only track runs made after the poll has been added. If you need to fetch older runs, then there is a management task which allows you to do this. The command takes two parameters:

1. The database id of the organization
2. A number of minutes, hours or days:

```
# fetches all runs for org #1 made in the last 45 minutes
$ ./manage.py fetchruns 1 --minutes=45

# fetches all runs for org #2 made in the last 6 hours
$ ./manage.py fetchruns 2 --hours=6

# fetches all runs for org #3 made in the last 2 days (48 hours)
$ ./manage.py fetchruns 3 --days=2
```

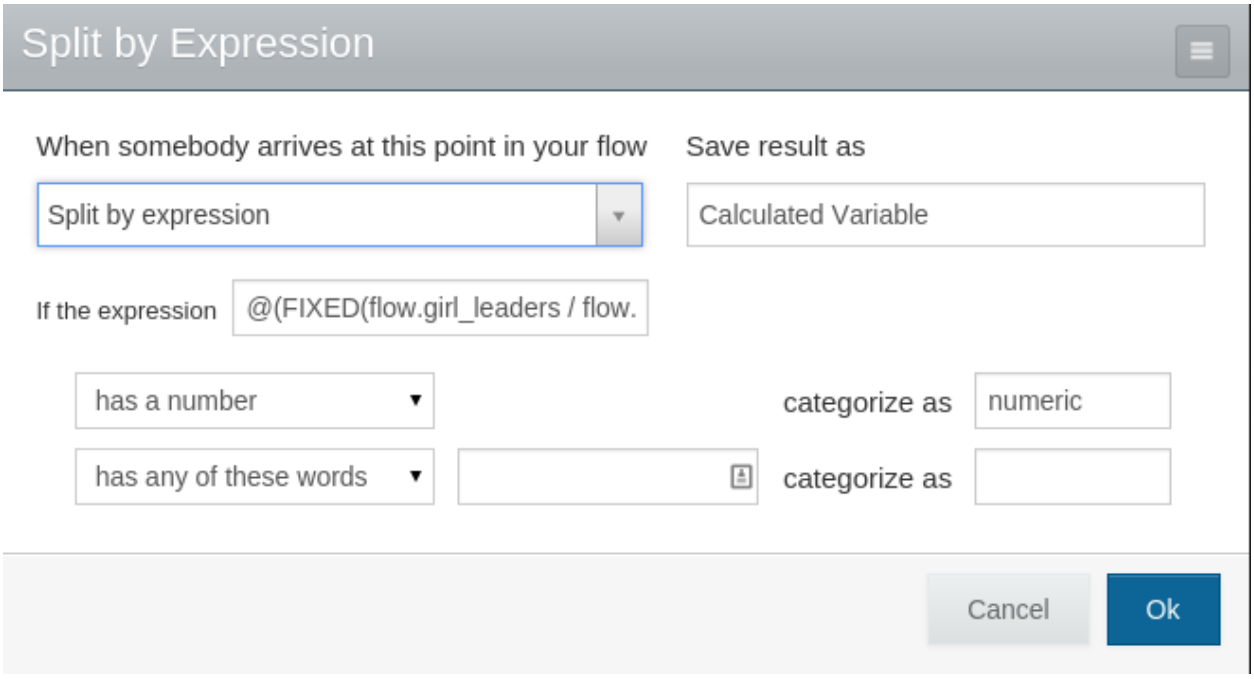
One should use this command with caution as it could potentially try to download a very high number of runs

2.5 Displaying Calculated Values on Tracpro

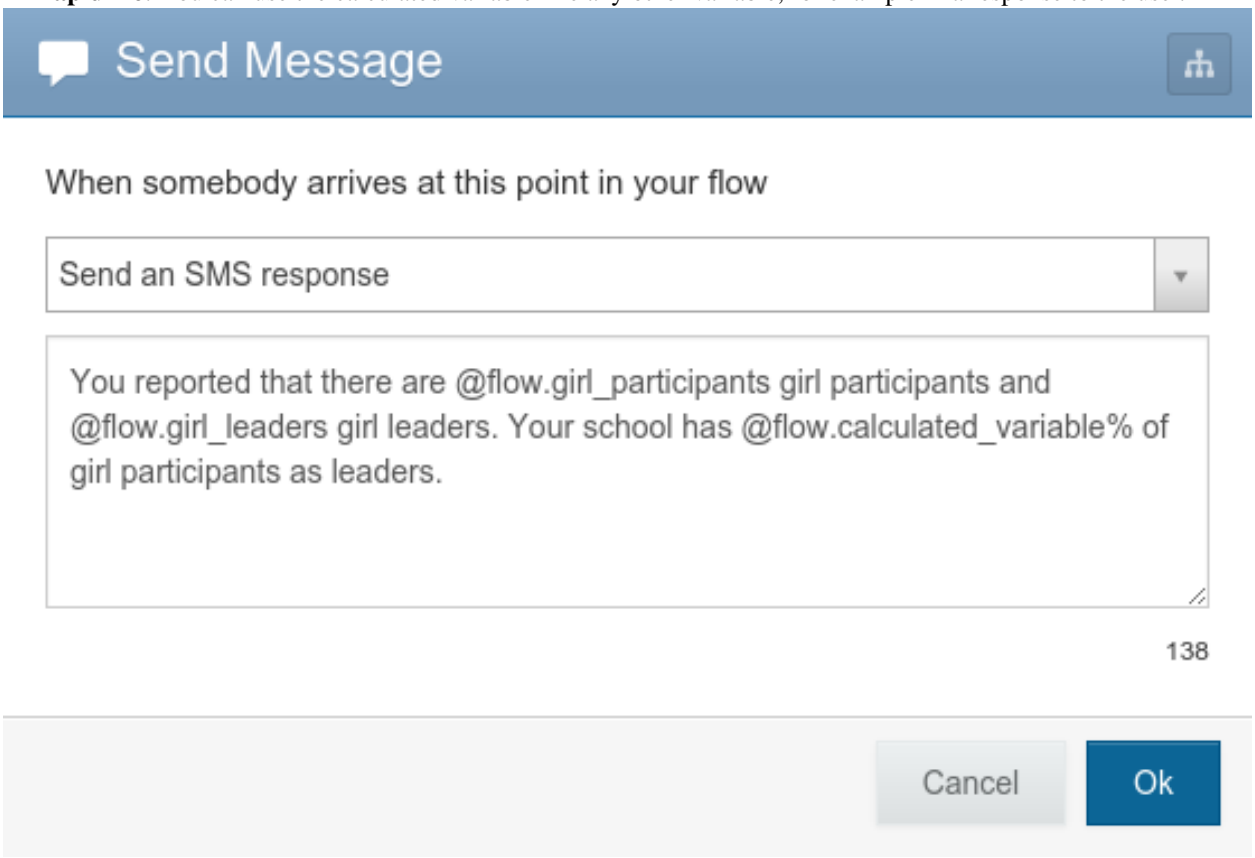
You can display values that are calculated from user responses on Tracpro. We reference an `example flow` that uses a calculated variable from results of a structured message form.

In RapidPro: After you have built your flow to collect variables that you wish to use for the calculation, add a "Split by Expression" step.

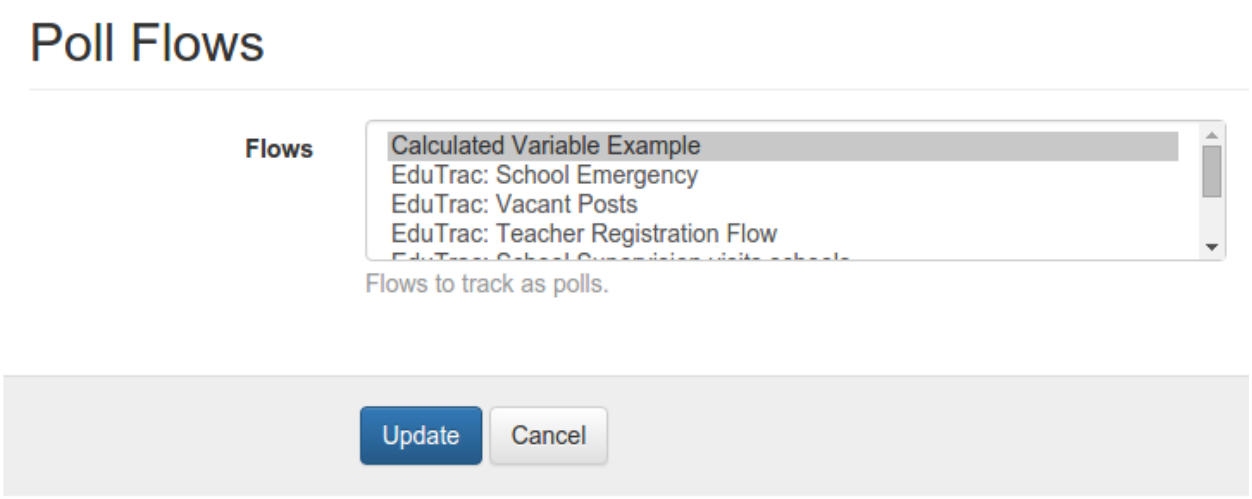
Save the result as a particular variable name, for example *Calculated Variable*. Write the expression to calculate the variable, for example *@(flow.girl_leaders / flow.girl_participants * 100)*. You have access to basic math, expressions, and variables.



In RapidPro: You can use the calculated variable like any other variable, for example in a response to the user.

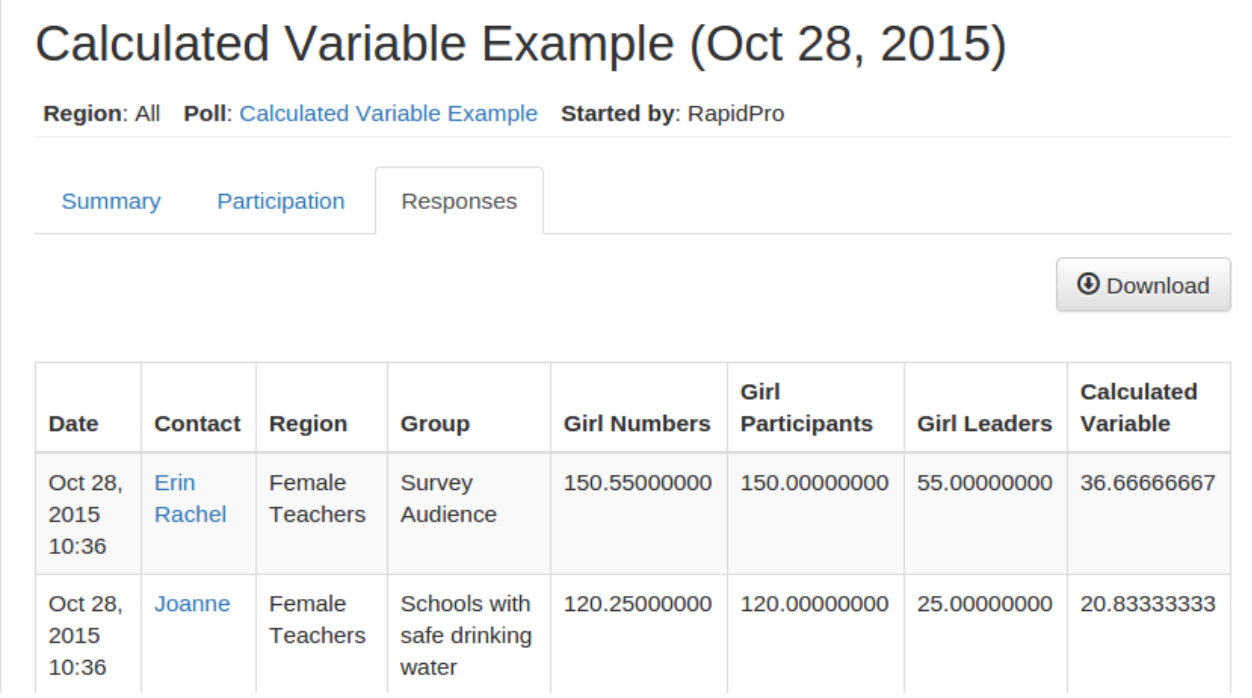


In TracPro: To view on Tracpro, ensure that your poll has been added through the Poll Flows chooser.



In TracPro: Your calculated variable will be available as a Question on Tracpro. You can view and use it exactly as you would any other Question.

- To view the numeric responses, including those for the calculated variable:
- View the Poll.
- Click on the "Dates" tab.
- Select the date of the poll you wish to see.
- Click on the "Responses" tab.



Date	Contact	Region	Group	Girl Numbers	Girl Participants	Girl Leaders	Calculated Variable
Oct 28, 2015 10:36	Erin Rachel	Female Teachers	Survey Audience	150.55000000	150.00000000	55.00000000	36.66666667
Oct 28, 2015 10:36	Joanne	Female Teachers	Schools with safe drinking water	120.25000000	120.00000000	25.00000000	20.83333333

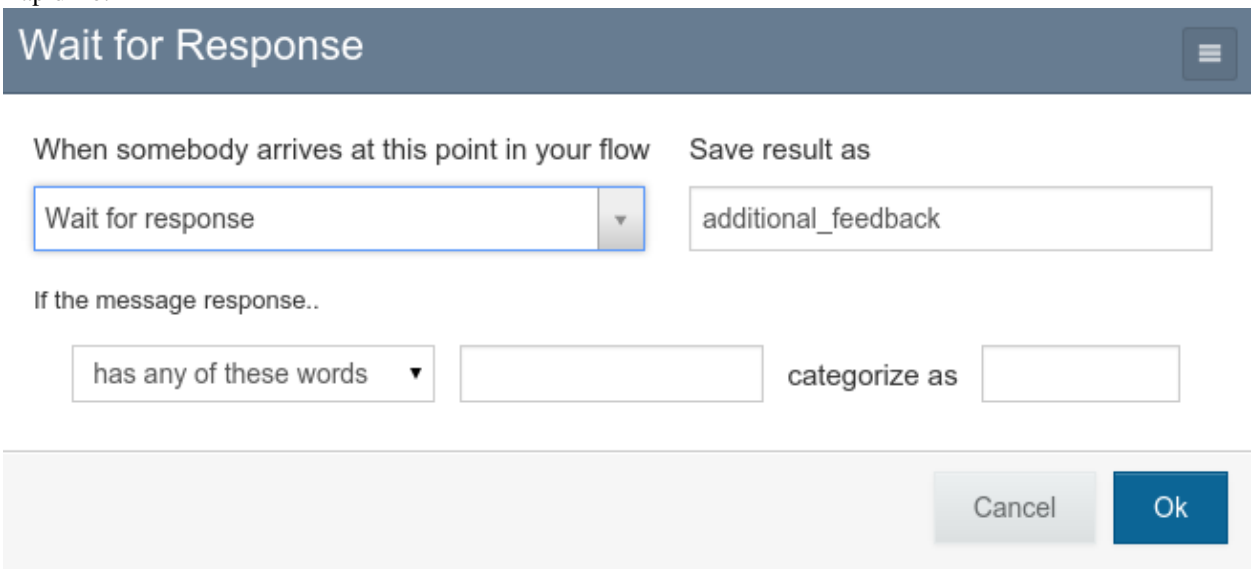
CHAPTER 3

Question types

TracPro questions are classified as either categorical or open-ended. Question type is determined by how the flow variable is set up on RapidPro.

3.1 Open-ended questions

To set up an open-ended question, do not change the default choices in the "If the message response..." section on RapidPro.

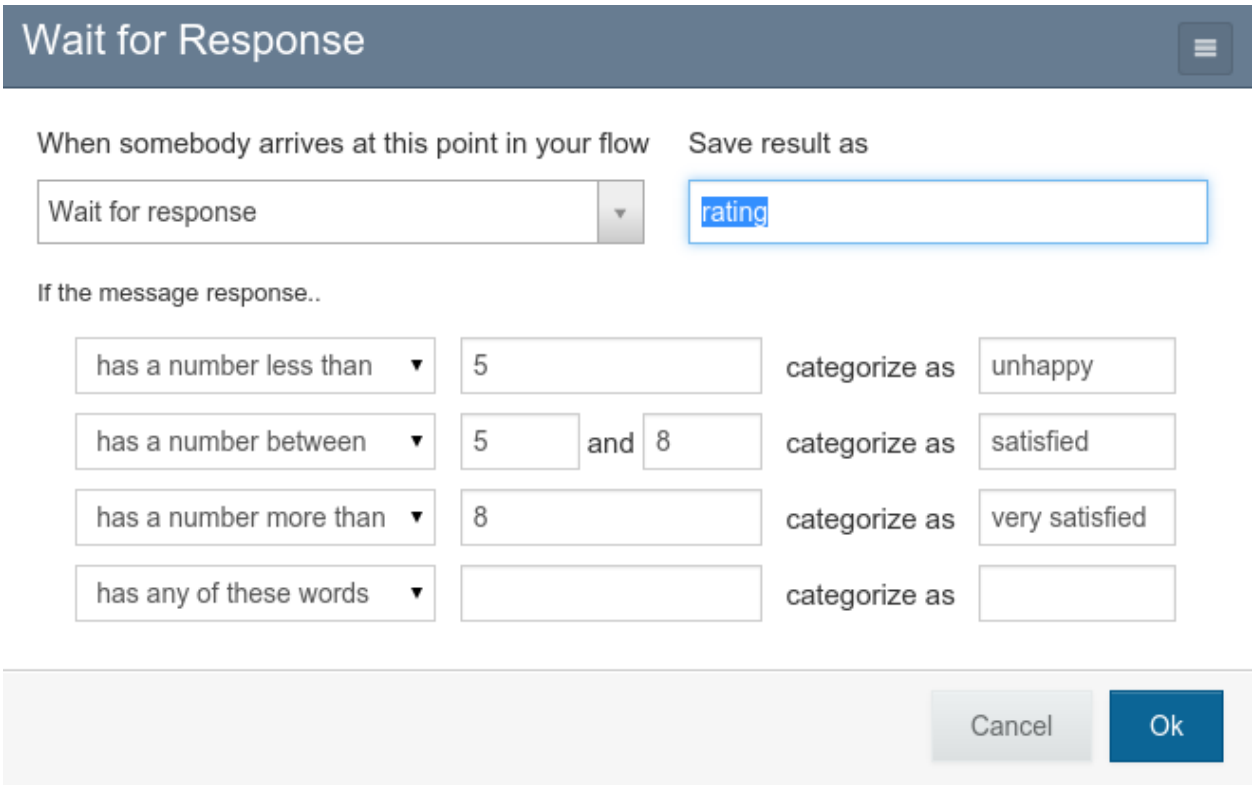


Your question will look like this on the flow editor:

Data for open-ended questions is displayed as a word cloud.

3.2 Categorical questions

To set up a categorical question, define categories for your responses on RapidPro.



Your question will look like this on the flow editor:

Data for categorical questions is displayed as a pie chart.

CHAPTER 4

Local Development Environment Setup

Use this guide to bootstrap your local development environment.

NOTE: These instructions are written with Ubuntu in mind. Some adjustments may be needed for Mac setup. To begin you should have the following applications installed on your local development system:

- Python 2.7
- pip >= 1.5
- virtualenv >= 1.10
- virtualenvwrapper >= 3.0
- Postgres >= 9.3
- git >= 1.7

1. Install the LESS CSS precompiler and the CoffeeScript JavaScript compiler using `npm`. First, install `nodejs` which comes with `npm`:

```
sudo apt-get install python-software-properties
sudo add-apt-repository ppa:chris-lea/node.js
sudo apt-get update
sudo apt-get install nodejs
```

   Then, use `npm` to install `less` and `coffee`:

```
sudo npm install less coffee-script -g
```

   NOTE: You may already have `less`, `coffee` or `npm` installed. Before running installation commands, use `which [program_name]` to see if the path to the program's executable is known.

2. Clone the repo and check out the `develop` branch:

```
$ git clone git@github.com:rapidpro/tracpro.git
$ cd tracpro
$ git checkout develop
```

3. Create a virtual environment using Python 2.7 and install the project requirements:

```
# Check that you have python2.7 installed
$ which python2.7
$ mkvirtualenv tracpro -p `which python2.7`
(tracpro)$ $VIRTUAL_ENV/bin/pip install -r $PWD/requirements/dev.txt
```

4. Create a local settings file:

```
(tracpro)$ cp tracpro/settings/local.example.py tracpro/settings/local.py
```

You may edit this file to make settings changes that are local to your machine. This file is listed in the .gitignore file and should never be checked into GitHub.

5. This project uses django-dotenv to manage environment variables. Configure the environment variable Django uses to locate the project settings file:

```
(tracpro)$ echo "DJANGO_SETTINGS_MODULE=tracpro.settings.local" >> .env
```

6. Create a Postgres database and run the initial migrate:

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

7. NOTE: TracPro uses Smartmin for permissions-based object scaffolding. If you make changes to a permission in GROUP_PERMISSIONS in tracpro/settings/base.py, you are required to migrate the database in order for that permission to take effect.

   (tracpro)$ python manage.py migrate

8. Background tasks. To run background tasks, you'll also need to start celery:

```
(tracpro)$ celery -A tracpro worker -B -l info
```

9. Subdomain Setup

   TracPro uses subdomains to determine which organization a user is currently accessing. For example, if you create an organization with the subdomain **testing**, you should configure that as an alias for localhost. On a UNIX-like system you would edit /etc/hosts as follows:

```
127.0.0.1   localhost. testing.localhost
```

10. RapidPro Integration

    The default development settings file connects to app.rapidpro.io.To integrate with a different RapidPro instance, either edit this file or create a new settings file.

11. Create Super User

    If creating a super user, be sure to select a valid password. TracPro enforces an 8 character minimum password.

12. Run the development server and navigate to localhost:8000:

```
python manage.py runserver
```

CHAPTER 5

Running Tests

TracPro uses Travis CI to automatically build and test code upon each push to GitHub.

For the Travis build to pass, all tests should pass, code coverage should be greater than 75%, and no Flake8 errors should exist.

To run the Django tests for all `tracpro` apps, run this command:

```
python manage.py test
```

You can also run the tests with coverage and check the code coverage results:

```
coverage run manage.py test
coverage report
```

To see HTML output of the coverage results (which is usually easier to read), run `coverage html` after running tests with coverage, then navigate to the file `htmlcov/index.html` (relative to the project root) in your browser.

To check PEP8 and pyflakes compliance:

```
flake8
```

Errors & their locations will be output; no output indicates success.

CHAPTER 6

Translation

This project uses the standard Django translation mechanisms.

6.1 What goes in version control?

Message files are located in the `locale/` directory. Files for each available language are in subdirectories named according to ISO language codes.

The `django.po` file in each language directory contains all translatable messages, plus the translation if available. Django reads translated messages at runtime from the `django.mo` file, which is the compiled version of the `.po` file. We commit both files to GitHub for ease of tracking changes and making reverts if needed.

6.2 Update translations

A developer should update message files when Python or template code with translatable messages is changed or when a new language is added. **NOTE:** To minimize noise, we generally update translation files just once per release cycle.

1. Create or update the message files:

```
python manage.py makemessages --ignore 'conf/*' \
                              --ignore 'docs/*' \
                              --ignore 'requirements/*' \
                              --no-location --no-obsolete -l en
```

2. Ensure that changes look reasonable using `git diff`. E.g., if translations have vanished, figure out why before proceeding.
3. Edit these files to add translations, if or when able.
4. Compile the message files:

```
python manage.py compilemessages
```

   If you get any errors due to badly formatted translations, work with your translators to fix the errors and start this process over.

5. Commit these changes to GitHub.

6.3 Add a new language

1. Add the language to the `LANGUAGES` setting. `tracpro/settings/base.py`.
2. If the language is written from right to left, add the language code to `RTL_LANGUAGES`.
3. Create an empty directory, named with the language's two-letter ISO code, in the `locale/` directory.
4. Update the translation files as described above.
5. Commit these changes to GitHub.

CHAPTER 7

Changelog

Tracpro's version is incremented upon each merge to master according to our *production deployment process*.

We recommend reviewing the release notes and code diffs before upgrading between versions.

7.1 v1.4.0 (released 2016-03-28)

Code diff: https://github.com/rapidpro/tracpro/compare/v1.3.3...v1.4.0

- Migrations to move RapidPro uuid unique constraint to unique_together with another model field (*org* for Contact, Region, and Group models; *pollrun* for Response)

7.2 v1.3.3 (released 2016-03-28)

Code diff: https://github.com/rapidpro/tracpro/compare/v1.3.2...v1.3.3

- Implement "backoff" for OrgTasks that fail
- Ensure that cache key timeout is set properly in OrgTask
- Do not use @task decorator on class-based task

7.3 v1.3.2 (released 2016-03-23)

Code diff: https://github.com/rapidpro/tracpro/compare/v1.3.1...v1.3.2

- Add debug logging for OrgTask

7.4 v1.3.1 (released 2016-03-23)

Code diff: https://github.com/rapidpro/tracpro/compare/v1.3.0...v1.3.1

- Fix formatting errors in this changelog
- Return *None* if *SoftTimeLimitExceeded* is raised during *OrgTask*
- Run *pipconflictchecker* on Travis builds
- Fail before running Travis tests if there are missing migrations

- Increase the *hard_time_limit* value for Org tasks

7.5 v1.3.0 (released 2016-03-22)

Code diff: https://github.com/rapidpro/tracpro/compare/v1.2.1...v1.3.0

Infrastructure

- Update to Django==1.8.11
- Update versions on many third-party packages (excluding forks)
- Updated the Caktus smartmin fork
- Serve library scripts and stylesheets from */static/libs/* rather than CDNs
- Ensure all test classes inherit from *TracProTest*, which ensures that critical features are mocked

Features & Bugfixes

- Fix email prefix on deployed environments
- Add *from __future__ import unicode_literals* to all files
- Only show responses from active contacts on charts for baseline, poll detail, and pollrun detail
- Don't abbreviate big numbers on charts (1,000,000 rather than 1M)
- Add user documentation about designing flows
- **Add *Boundary* model to *tracpro.groups***
  - Sync with RapidPro
  - Add endpoint to retrieve all boundaries for an Org
- Add *boundary* foreign key to *Region* and allow setting the *boundary* on the Region list page
- Add contact data field filters to PollRun detail page & pass applicable filters to PollRun detail page when clicking on a data point on the Poll detail page.
- Store ruleset on the *Question* model
- Add ability to categorize arbitrary (numeric) values
- Display results on a map

7.6 v1.2.1 (released 2016-03-21)

Code diff: https://github.com/rapidpro/tracpro/compare/v1.2.0...v1.2.1

- Fix EMAIL_HANDLER
- Add *django* logger
- Prevent Celery from hijacking the root logger

7.7 v1.2.0 (released 2016-03-14)

Code diff: https://github.com/rapidpro/tracpro/compare/v1.1.1...v1.2.0

- **Settings changes:**
  - Update *LOGGING* to reflect sending logs to *syslog*
  - Utility for grabbing settings from the environment
  - Utility for falling back to Django default settings
  - Email configuration
  - Remove unused *HOSTNAME* setting
  - Misc. settings tweaks related to deployment.

7.8 v1.1.1 (released 2016-03-01)

Code diff: https://github.com/rapidpro/tracpro/compare/v1.1.0...v1.1.1

- Updated to Django==1.8.10 from Django==1.8.6
- Send Celery task error emails.
- Limit InboxMessages fetch to the past 7 days.
- Use relativedelta where possible.
- Update 404 page template.

7.9 v1.1.0 (released 2016-02-24)

Code diff: https://github.com/rapidpro/tracpro/compare/v1.0.4...v1.1.0

Many changes, including:

- Break out deployment-related assets into a private repo.
- **Update Celery task structure.**
  - **Note:** Existing tasks are probably very backed up. After deploy, purge all existing tasks (see Celery FAQ).
- Chart enhancements on Poll detail and PollRun detail pages.
- Filters on Recent Indicators, Poll detail, and PollRun detail pages.

7.10 v1.0.4 (never released)

Code diff: https://github.com/rapidpro/tracpro/compare/v1.0.3...v1.0.4

- Update versions of Celery-related packages.

7.11 v1.0.3 (released 2015-11-30)

Code diff: https://github.com/rapidpro/tracpro/compare/v1.0.2...v1.0.3

- Bug fix for clearing spoof data. See #100.
- Release notes added for ReadTheDocs builds

7.12 v1.0.2 (released 2015-11-25)

Code diff: https://github.com/rapidpro/tracpro/compare/v1.0.1...v1.0.2

- Don't paginate results on responses CSV export.
- Show participant count in participant column on PollRun ByPoll page.

7.13 v1.0.1 (released 2015-11-25)

Code diff: https://github.com/rapidpro/tracpro/compare/v1.0.0...v1.0.1

- Updated contact sync to run every 30 minutes, rather than every 5.

7.14 v1.0.0 (released 2015-11-19)

Code diff: https://github.com/rapidpro/tracpro/compare/v0.0.51...v1.0.0

- Add documentation to ReadTheDocs.
- **Upgrade version requirements.**
  - **Note:** Due to a change in structure for *django-celery*, you will need to run *python manage.py migrate djcelery –fake-initial* before running new migrations.
- Add *prod_db_to_staging* Fabric command.
- Fix *hostname* in *manage_run* Fabric command so that it now runs without error.
- Require that source is updated before updating pip requirements during deploy.
  - **Note:** Pip requirements were sometimes being updated before the source code was updated. If you have this issue before updating to v1.0.0, run the deploy again to solve.
- **Add deadsnakes Python 2.7 to deploy environment.**
  - **Note:** An SSL dependency requires Python 2.7.9 or greater. If your deployment is using a lower version, destroy the virtual environment before your next deploy so that it is rebuilt.
- Add org config option to show/hide spoof data. See #92.
  - **Note:** A migration sets the default to False for all orgs except "Caktus".
- Fix unicode bug when setting a Contact DataField value. See #88.
- Use `django.utils.dateparser` rather than `dateutil` when parsing datetimes for DataFields. See #88.
- Fix org languages bugs. See #91.

CHAPTER 8

Deployment Process

8.1 Release to staging

1. Ensure that your work has been pushed to GitHub.
2. On your local machine, check out the branch that you wish to deploy.
3. By default, the `develop` branch is deployed to staging (regardless of which branch you've checked out locally). If you wish to deploy a different branch, you will need to edit your local copy of conf/pillar/staging.sls to use a different branch name.
4. Run the deploy:

```
fab staging deploy
```

8.2 Reset staging environment

These steps will restore the production database and code states to the staging environment.

1. Copy the production database to staging:

```
fab prod_db_to_staging
```

2. On your local machine, check out the master branch:

```
git checkout master
```

3. Edit your local copy of conf/pillar/staging.sls to specify the `master` branch for deployment.
4. Deploy the `master` branch to staging:

```
fab staging deploy
```

5. Undo the change to conf/pillar/staging.sls.

8.3 Release to production

1. **Before** merging changes to `master`, the complete deployment process should be tested on staging.
    - *Restore* production data, media, and code to the staging machine.

- Check out the `develop` branch locally and deploy it to staging:

```
fab staging deploy
```

NOTE: If you are deploying a hotfix branch, you will need to edit your local copy of conf/pillar/staging.sls accordingly.

- Confirm that changes are behaving as expected, and troubleshoot the deploy process until you are confident.

2. Finalize the release number.
   - Update the version number according to semantic versioning, and change the version state to `"final"`. Version must be updated in `tracpro/__init__.py` and `docs/source/conf.py`.
     - The micro version is incremented with backwards-compatible bugfixes.
     - The minor version is incremented with backwards-compatible features.
     - The major version is incremented with incompatible changes.
   - Update the release notes (including notes about one-off deployment requirements, if needed) and add the current date as the release date.
   - Commit these changes and push to GitHub.
3. Merge all changes to the `master` branch, ensure that these changes are pushed to GitHub, and confirm that the Travis build has passed.
4. Check out the updated `master` branch locally, and create a release tag:

```
git tag -a vX.Y.Z -m "Released YYYY-MM-DD" && git push origin --tags
```

5. Copy the release notes to the GitHub releases interface.
6. Run the production deploy:

```
fab production deploy
```

7. Merge the `master` branch into `develop`:

```
git checkout develop
git merge origin/master
git push origin develop
```

8. On the `develop` branch, increment the micro version and change the code state to `"dev"`. Commit these changes and push to GitHub.
9. Run ReadTheDocs builds for the new release & the latest develop.

CHAPTER 9

Fabric commands

TracPro uses the Caktus project template. See the server documentation to learn more about server architecture and provisioning.

Before using fabric commands, follow all of the steps in the developer setup docs to ensure that you have all of the required credentials.

You can get a full list of Fabric commands by running `fab --list`, or read through the source at fabfile.py.

9.1 Copying data for local development

To copy the database and media files from a remote server to your local development environment:

```
fab [production|staging] reset_local_db
```

NOTE: Migrations must be run (`django-admin migrate`) if the local branch has migrations that have not yet been run on the remote server.

9.2 Copying production data to staging

To copy the production database to staging:

```
fab prod_db_to_staging
```

NOTES: Migrations must run (`fab staging manage_run:"migrate"`) if the branch that is deployed on staging has new migrations that have not been run on production.

9.3 Running commands on the server

To run a management command on a remote server:

```
fab [production|staging] manage_run:"command_name -arg --option param1 param2"
```

Use the `manage_shell` alias to run a Python shell on a remote server:

```
fab [production|staging] manage_shell
```

CHAPTER 10

Server Provisioning

10.1 Overview

Tracpro is deployed on the following stack.

- OS: Ubuntu 14.04 LTS
- Python: 2.7
- 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.

10.2 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
```

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.

10.3 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.

10.4 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.

10.5 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.

10.6 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
```

10.7 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`

10.8 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 `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.

10.9 Optional Configuration

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

10.9.1 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.

10.9.2 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 `tracpro/__init__.py` so that you can run the worker via:

```
celery -A tracpro worker
```

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

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

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

You will also need to add the `BROKER_URL` to the `tracpro.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 tracpro_<environment> and the vhost will be named tracpro_<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.

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

CHAPTER 11

Server Setup

11.1 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>*.

11.2 Layout

Below is the server layout created by this provisioning process:

```
/var/www/tracpro/
    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.

11.3 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 12

Vagrant Testing

12.1 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.

12.2 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 tracpro.settings.staging import *

# Override settings here
DATABASES['default']['NAME'] = 'tracpro_local'
DATABASES['default']['USER'] = 'tracpro_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.

12.3 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.