frb_cand

Release 1

ADACS

Jan 04, 2023

USING THE WEB APPLICATION:

CHAPTER

ONE

UPLOADING FRB EVENTS

1.1 Running the Upload Script

To upload your FRB events and follow up measurements you can use this python script. To use it you must set the environment variables *FRB_USER* and *FRB_PASS* which is your username and password for the FRB web app account.

For the first detection/measurement of the FRB event, you can upload the radio measurement and observation data with a command like the following:

```
python upload_cand.py --first --radio_yaml radio_example.yaml --observation_yaml␣
↪observation_example.yaml
```

Which will output an ID like so:

```
3
```

This should be recorded and used for future measurement updates (through post-processing and optical follow up).

To upload further measurements, use the update option like so:

```
python upload_cand.py --update 3 --radio_yaml radio_example.yaml
```

Note that you don't need the observation YAML after the first detection.

1.2 Radio Measurement YAML Format

Here is an example of what the radio measurement YAML can look like

```
{
    # Only used for first detection/measurement
    "time_of_arrival": "2017-11-17T12:21:38.87",
    "repeater": true,
    "search_path": "example_search.png",
    "image_path" : "example_image.png",
    "histogram_path": "example_histogram.png",

    # Required
    "dm": 411.0,
    "dm_err": 1.0,
    "ra": 77.01461542,
```

(continues on next page)

(continued from previous page)

```
    "ra_err": 0.05,
    "dec": 26.06069556,
    "dec_err": 0.05,
    "sn": 50,
    "width": 5,
    "flux": 35,
    "flux_err": 3,
    "source": "MB",
    "version": "v1.0",

    # Optional
    "fluence": 45,
    "fluence_err": 5,
    "dmism": 123.16007817568256,
    "rm": -613.0,
    "rm_err": 2.0,
    "cosmo": "Planck18",
    "eellipse": {
        "a": 0.004,
        "b": 0.004,
        "cl": 68.0,
        "theta": 0.0
    },
    "z": 0.0982,
}
```

Each of the keys:

"time_of_arrival": *str*, optional
The time of arrival of the FRB in the format "%Y-%m-%dT%H:%M:%S.%f", eg. "2017-11-17T12:21:38.87"

"repeater": *boolean*, optional
Is the FRB a repeater (true or false)?

"search_path": *str*, optional
The path to the search image

"image_path"
[*str*, optional] The path to the radio image

"histogram_path": *str*, optional
The path to the histogram image

"dm"
[*float*] The dispersion measure of the FRB in pc / cm^3

"dm_err"
[*float*] The error of the dispersion measure of the FRB in pc / cm^3

"ra": *str*
The Right Acension of the candidate in degrees

"ra_err": *str*
The error of the Right Acension of the candidate in degrees

"dec": *str*
The Declination of the candidate in degrees

"dec_err": *str*
The error of the Declination of the candidate in degrees

"sn": *float*
The signal-to-noise ration of the candidate

"width": *float*
The width of the candidate pulse in ms

"flux": *float*, optional
The flux density of the event in Jy

"flux_err": *float*, optional
The error of the flux density of the event in Jy

"source": *str*
The source (telescope pipeline) of the measurements, should be either MB (Multi-Beam) or HT (High-Time resolution)

"version": *str*
The version of the "source" software

"fluence": *float*, optional
The fluence of the event in Jy ms

"fluence_err": *float*, optional
The error of the fluence of the event in Jy ms

"dmism"
[*float*, optional] The estimated amount of the dispersion measure that is contributed by the interstellar medium in pc / cm^3

"rm": *float*, optional
The Rotation Measure of the candidate in rad / m^2

"rm_err": *float*, optional
The error of the Rotation Measure of the candidate in rad / m^2

"cosmo": *str*, optional
The cosmological model used for cosmological calculations, eg. "Planck18"

"eellipse": *object*, optional
The error ellipse object which has the following keys within it

> **"a": *float***
> The width of the ellipse in degrees
>
> **"b": *float***
> The height of the ellipse in degrees
>
> **"cl": *float*, optional**
> The confidence level of the error ellipse in percent. Default 68.0
>
> **"theta": *float***
> The angle in degrees from North clockwise

"z": *boolean*, optional
The redshift of the candidate

1.3 Observation YAML Format

Here is an example of what the observation YAML can look like

```
{
    "beam_semi_major_axis": 0.2,
    "beam_semi_minor_axis": 0.3,
    "beam_rotation_angle": 45,
    "sampling_time": 0.1,
    "bandwidth": 300,
    "nchan": 3000,
    "centre_frequency": 1400,
    "npol": 2,
    "bits_per_sample": 8,
    "gain": 3,
    "tsys": 50,
    "backend": "Multibeam",
    "beam": 1,
}
```

"beam_semi_major_axis": *float*
The beam semi major axis in arcminutes.

"beam_semi_minor_axis": *float*
The beam semi minor axis in arcminutes.

"beam_rotation_angle": *int*
The beam rotation angle in degrees, clockwise from North.

"sampling_time": *float*
The duration of each sample in ms.

"bandwidth": *float*
The bandwidth in MHz.

"nchan": *int*
The number of frequency channels.

"centre_frequency": float`
The centre frequency in MHz.

"npol": *int*
The number of antena polarisations.

"bits_per_sample": *int*
The size in bits of each sample.

"gain": *float*
The gain of telescope in K/Jy.

"tsys": *float*
The system temperature in K.

"backend": *string*
The name of the telescope backend being used ("Multibeam" for example).

"beam": *int*
The beam number for multi beam receivers.

CHAPTER

TWO

DATABASE INSTALLATION

2.1 Dependancies

For Ubuntu or Debian Linux:

```
sudo apt-get update
sudo apt-get install postgresql postgresql-contrib libpq-dev python3-dev graphviz␣
↪python3-pip
```

Then install the python requirements (recommended in its own virtual environment) using:

```
pip install -r frb_cand/requirements.txt
```

2.2 Environment Variables

To run the web application, you will need to set the following environment variables:

Variable	Description
DB_USER	Postgres user name which you will set in the next section.
DB_PASSWORD	Postgres password which you will set in the next section.
DB_SECRET_KEY	Django secret key. Here is a description of how to generate one.
SYSTEM_ENV	Set this either to 'PRODUCTION' to turn off debug and enable CSRF_COOKIE_SECURE, or 'DEVELOPMENT' to turn on debug
UPLOAD_USER	A username of an account that will be used by upload_xml.py to upload VOEvents
UPLOAD_PASSWORD	The password of the upload user

2.3 Start the Postgres Database

The following commands will set up the Postgres database for the web app. Replace $DB_USER and $DB_PASSWORD with the environment variable values.

```
sudo -u postgres psql

CREATE DATABASE frb_cand_db;
```

(continues on next page)

(continued from previous page)

```
CREATE USER $DB_USER WITH ENCRYPTED PASSWORD '$DB_PASSWORD';

ALTER ROLE $DB_USER SET client_encoding TO 'utf8';
ALTER ROLE $DB_USER SET default_transaction_isolation TO 'read committed';
ALTER ROLE $DB_USER SET timezone TO 'UTC';
```

2.4 Setup database for the first time

Run the following commands from the webapp_tracet subdirectory so Django can setup up the database structure and upload defaults

```
python manage.py migrate
python manage.py migrate --run-syncdb
```

2.5 Create a superuser

These commands will set up a superuser account.

```
python manage.py createsuperuser
```

2.6 Delete Postgres Database

Only do this is you want to restart the database!

To delete the database use the following commands

```
sudo -u postgres psql

DROP DATABASE frb_cand_db;
CREATE DATABASE frb_cand_db;
```

You will then have to recreate the database using the commands in *Setup database for the first time*

CHAPTER

THREE

WEB APPLICATION INSTALLATION

The following are instructions on how to setup up your nimbus instance for the first time. If you have already done this you can skip to *Starting the server*.

3.1 Opening the Nimbus Instance Firewall

Once you've set up the instance you need to open the firewall

https://support.pawsey.org.au/documentation/display/US/Allow+HTTPS+Access+To+Your+Instance

Then make a costum tcp rule for ports 80 and 443, should look like this

Displaying 4 items

Direction	Ether Type	IP Protocol	Port Range	Remote IP Prefix	Remote Security Group	Desc
Egress	IPv4	Any	Any	0.0.0.0/0	-	-
Egress	IPv6	Any	Any	::/0	-	-
Ingress	IPv4	TCP	80 (HTTP)	0.0.0.0/0	-	-
Ingress	IPv4	TCP	443 (HTTPS)	0.0.0.0/0	-	-

Displaying 4 items

Then follow this guide to check things step by step

https://uwsgi-docs.readthedocs.io/en/latest/tutorials/Django_and_nginx.html

The following is examples of how I got it to work.

3.2 Goal 1: IP as URL

First try and get it to work with the nimbus IP as the URL. From directory containing manage.py run the command:

```
uwsgi --socket frb-classifier.sock --module frb_cand.wsgi --chmod-socket=666
```

and nginx should look like this

```
upstream django {
   server unix:///home/ubuntu/FRB_candidates_app/frb_cand/frb-classifier.sock;
}

server {
   listen 80;
   server_name <IP>;
   charset     utf-8;

   # max upload size
   client_max_body_size 75M;

   location /static {
      alias /home/ubuntu/FRB_candidates_app/frb_cand/static_host;
   }

   # Finally, send all non-media requests to the Django server.
   location / {
      uwsgi_pass  django;
      include     /home/ubuntu/FRB_candidates_app/frb_cand/uwsgi_params;
   }

}
```

and make sure the IP is in allowed hosts in settings.py:

```
ALLOWED_HOSTS = ['127.0.0.1', 'localhost', '<IP>']
```

Check if the works by using the IP as a URL in your browser.

3.3 Perimission errors

If you get a *(13: Permission denied)* error in the nginx logs here is a helpful fix

https://stackoverflow.com/questions/25774999/nginx-stat-failed-13-permission-denied

3.4 Static files errors

If it's not finding the static files then setup the setting.py like this

```
STATIC_URL = '/static/'
STATICFILES_DIRS = (
   os.path.join(BASE_DIR, "static/"),
)
STATIC_ROOT = os.path.join(BASE_DIR, "static_host/")
```

then run

```
python manage.py collectstatic
```

and update the nginx to

```
location /static {
   alias /home/ubuntu/FRB_candidates_app/frb_cand/static_host;
}
```

3.5 Try a simple domain

Grab a free subdomain from https://www.duckdns.org/domains that points to your ip then update the url in nginx's sevename, and ALLOWED_HOSTS in settings.py

3.6 Getting a ssl certificate

Here are instructions on generating a ssl certificate

https://certbot.eff.org/instructions?ws=nginx&os=ubuntufocal

CHAPTER

FOUR

RUNNING SERVER

4.1 Checking for errors and inspecting logs

nginx errors are in

```
tail -f cat /var/log/nginx/error.log
```

All commands assume you're in the frb_cand sub directory. You can see the output of the server with

```
tail -f uwsgi-emperor.log
```

4.2 Starting the server

Start the uwsgi server with

```
uwsgi --ini frb_cand_uwsgi.ini
```

This will run in the background and the following sections describe how to restarting and stopping the server.

4.3 Restarting the server

```
kill -HUP `cat /tmp/project-master.pid`
```

4.4 Stopping the server

```
uwsgi --stop /tmp/project-master.pid
```

4.5 Installing updates

If the updates are small normally something as simple as the following will suffice:

```
git pull
kill -HUP `cat /tmp/project-master.pid`
```

Larger updates may need a combination of the following commands

```
git pull
# Stop server
uwsgi --stop /tmp/project-master.pid
# Check for new dependent software
pip install -r requirements.txt
# Check for new static files
python manage.py collectstatic
# Make any required changes to the backend database
python manage.py makemigrations
python manage.py migrate
# Start server
uwsgi --ini frb_cand_uwsgi.ini
```

CHAPTER

FIVE

SOFTWARE DEVELOPER DOCUMENTATION

5.1 Uploading data

Users will upload data to the database with the *upload_cand.py* script, which parses input YAML files and puts the data into the relevant tables. The triggered steps are handled using `signals.py`, which will be explained in the next sections.

5.2 Slack Integration

Each time a new `FRBEvent` is created, the `slack_trigger` and `slack_event_post` functions in `signals.py` will be triggered and send off a Slack message describing the event. This is done in the `slack_event_post` view, which makes a block for each image and two response buttons.

The Slack App that handles this is frb_cand which posts to the private channel `nick_frb_cand_testing` currently. This is sent to slack using a webhook URL, which should be set in the settings.ini and changed here (if you have permissions).

This could be changed in production, or making a separate app for production may be more manageable. The required permissions are

- Post messages to specific channels in Slack
- View basic information about public channels in a workspace
- Send messages as @frbcand (or whatever the new app name is)
- Post messages to a private group

The buttons will return a JSON dump to the web app through the `slack_get_rating` view. The URL for the app is set within the Interactivity & Shortcuts settings page for the app (currently https://frb-classifier.duckdns.org/slack_get_rating/). The JSON is parsed, and the rating is recorded in the database and with a slack message.

5.3 Transient Name Server Integration

The TNS has a production version (https://www.wis-tns.org/) and a testing version (https://sandbox.wis-tns.org/) of the site. We currently only use the testing sandbox version. There is some documentation but it is lacking for the FRB API.

We only submit the first FRB radio measurement to the TNS. It submits the FRBEvent with the `submit_frb_to_tns` view, which dumps a JSON the TNS CRAFT_bot and then waits for a response that contains the transient name. We then record this name in the database.

You should have access to the CRAFT_bot as long as you have access to the CRAFT TNS group. You can request an invite to the CRAFT TNS group from Ryan Shannon.

Sometimes the API stops working, but updating the API key fixes it. Go to the Edit CRAFT_bot page and tick "Create new API Key" and click save. Copy the key it outputs into the `TNS_API_KEY` in the `settings.ini` and restart the server.

5.4 VOEvent Integration

Currently, we only locally (not to a broker) submit VOEvents for the first radio detection (no follow-ups or event withdrawals). `make_voevent` in `signals.py` creates the VOEvent using voevent-parse and using this template. It then submits the event using `comet-sendvo` and records it in the database.