# Introduction

This application note describes the CANARIE Research Software Reference application. It demonstrates a simple implementation of the NEP-RPI API as described in the documents entitled “Research Service Support for the CANARIE Registry and Monitoring System”, revision 6 and “Research Platform Support for the CANARIE Registry and Monitoring System”, revision 2. The aim of this document is explain the design of the application, how to use it and some notes on deploying it in a production environment.

The service is written using the following technologies and frameworks:

* Python[[1]](#footnote-1) - programming language
* django[[2]](#footnote-2) - web development framework
* django-rest-framework[[3]](#footnote-3) – REST frame work for django
* celery[[4]](#footnote-4) - task queue
* requests[[5]](#footnote-5) – Python HTTP library

The reference application functionality has deliberately been kept very simple. It consists of a service that count the number of times it is invoked and a platform that can regularly invoke the service. Both the platform and service implement the appropriate CANARIE Research Software APIs.

# Application Design

The service follows the layout of a standard django project with the following structure:

reference

integration

project

canarie\_platform

canarie\_service

project

util

logs

The code is contained within the reference/project directory and subdirectories. The main django management script is contained at this root level.

manage.py – Allows starting of the development server, running unit tests and accessing the Python iterative console with django support configured.

The subdirectory reference/project/project directory contains all of the site wide configuration and deployment details. The main files here are:

settings.py and settings-prod.py – The configuration settings for the default development and production deployment respectively.

urls.py – Which contains the basic site URL routing

wsgi.py and wsgi-prod.py – Contains the configuration to allow apache’s mod\_wsgi module to server the site. The configuration for the django development server is in wsgi.py and the configuration for apache in a production deployment is in wsgi-prod.py

The main site code for the application is contained in the reference/project/canarie\_service and reference/project/canarie\_platform directories. The main files used for each of these are:

defaults.py – Contains some default settings for the component.

models.py – Contains the database model for the application. This is used by django to configure the database schema and by the application to connect with the database.

serializers.py – Contains classes to serialize the Python data representations into the JSON format required by the API.

tests.py – Unit tests to test the functionality of the application. In this project all of the unit tests are contained in this one file.

views.py – This is the main application code for the applications, handles rendering of the API responses.

urls.py – Configures the URL routing to view based on regular expression rules.

templates/ - This subdirectory contains the templates for the various HTML pages rendered by the view.

The reference/project/canarie\_platform directory also contains some additional files.

tasks.py – The celery task that is used to run the polling.

utility.py – Some utility functions used in by the platform.

The reference/project/util directory contains some shared utilities for both the service and platform.

At a high level the basic flow of a request through this system is as follows. The requested URL is compared to the regular expressions in reference/project/project/urls.py. It is then passed to the urls.py in the appropriate application where is compared to the URLs configured there. If a match is made, the URL is passed to the appropriate handler in the views.py file.

In both cases the view script, view.py, contains all of the actual application processing. This file is divided into three sections. The first section contains all of the methods required to handle the CANARIE RPI API, the two most notable methods being ‘info’ and ‘stats’ which are annotated with and api\_view from the django-rest-framework API to help with processing REST calls. These methods check to see what ‘Accept’ header has been set in the request. If it is ‘application/json’ then a JSON representation of the data is returned; otherwise an HTML representation is rendered from the appropriate template. The rest of the API methods either redirect to an appropriate URL or return a rendered template directly.

The second section of file view.py contains methods to run the sample applications that the API is reporting on. The sample applications have deliberately been kept a very simple in order to just outline the use of the API.

For the service the main app method renders a page template with the current statistic data and contains a form for incrementing the count and resetting it. The ‘usage’ method is called from the form submit, then depending on the form fields that have been set the it calls method to either increment the counter or reset it and store the last update time. There are also two methods, ‘add’ and ‘reset’ to expose this functionality programmatically. The final method in this section allows the service ‘info’ to be updated via a REST PUT.

The platform renders a simple control for that allows starts and stops a random polling to the service. The page displays the number of interactions with the platform, that last time the statistics were reset, whether it is currently polling the service or not and the currently configured min and max second values for each subsequent poll.

The third section contains utility methods to support the API and application.

# How to use the service

This section will explain how to use the service application.

## Main application

Browse to:

<server-base-url>/reference/service/app

This page will display the current usage stats. Clicking on the ‘Count’ button will increment the usage count by one. Clicking on ‘Reset’ will reset the counter to ‘0’ and set the last reset time to the current server time in UTC. In either case the page will be re-rendered and the current values displayed.

The following REST calls are also available to interact with the application. When calling the REST method the ‘Accept’ HTTP header needs to be set to ‘application/json’ in order to make the request.

## Add

PUT <server-base\_url>/reference/service/add – to increment the usage counter programmatically

This will return a JSON object with the new values, in the form detailed in the CANARIE Research Middle ware API for ‘stats’, i.e.

{

“invocations” : “<invocation-count>”,

“lastReset” : “<last-reset-time>”

}

## Reset

PUT <server-base\_url>/reference/service/reset – to reset the usage counter and last reset time programmatically.

This will return the same JSON as for ‘add’ which contains a ‘invocations’ value of ‘0’ and the new ‘lastReset’ time in ISO8601 as detailed in the in the CANARIE Research Middleware API document.

## SetInfo

PUT <server-base\_url>/reference/service/setinfo

This expects a JSON payload in the form of the ‘info’ object detailed in the CANARIE Research Middle ware API document. i.e.

{

“name” : “<new-name>”,

“synopsis” : “<new-synopsis>”,

“version” : “<new-version>”,

“institution” : “<new-institution>”,

“releaseTime” : “<new-release-time>”,

“researchSubject” : ”<new-research-subject>”,

“supportEmail” : “<new-support-email>”,

“category” : “<new-category>”,

“tags” : [“<new-tag-1>”, “<new-tag-2>” …]

}

The info settings for the service will be updated with the new values and it will return a JSON object containing these new settings in the same format.

Release time should be in the format ‘%Y-%m-%dT%H:%M:%SZ’.

## General usage

The application will also respond to the URLs detailed in “Research Service Support for the CANARIE Registry and Monitoring System” with the correct data.

# How to use the platform

This section explains how to use the platform.

## Main application

Browse to:

<server-base-url>/reference/platform/app

The will display a simple page that allows you to see the current stats about the platform and along with the current service polling state. There are buttons to start and stop the polling as well as a button to reset the current usage count. Both starting and stopping the service will result in a usage increase.

## Update

POST <server-base-url>/reference/platform/update

With a form encoded payload in the form “action” : “<action>” where <action> is one of ‘start’, ‘stop’ or ‘reset’.

This will either start or stop polling the service depending on the parameter supplied or reset the usage statistics.

A simple example of this using the python requests library is:

reference\_url = 'http://localhost:8000'

headers = {'accept': 'application/json'}

payload = {'action': 'start'}

r = requests.post('{}/reference/platform/update'.format(reference\_url), headers=headers, data=payload)

## SetInfo

PUT <server-base\_url>/reference/service/setinfo

This expects a JSON payload in the form of the ‘info’ object detailed in the CANARIE Research Middle ware API document. i.e.

{

“name” : “<new-name>”,

“synopsis” : “<new-synopsis>”,

“version” : “<new-version>”,

“institution” : “<new-institution>”,

“releaseTime” : “<new-release-time>”,

“researchSubject” : ”<new-research-subject>”,

“supportEmail” : “<new-support-email>”,

“tags” : [“<new-tag-1>”, “<new-tag-2>” …]

}

The info settings for the service will be updated with the new values and it will return a JSON object containing these new settings in the same format. Release time should be in the format ‘%Y-%m-%dT%H:%M:%SZ’.

# Admin console

Many of the parameters and settings can also be modified using the django administration console. This can be accessed at:

<server-base\_url>/admin

The admin credentials are those supplied when the django database is created. More user credentials can be created if needed. This admin console allows administration of both service and platform data. The database is populated with default data the first time that data is accessed. So, for example, browsing to the platform/info page may be required after installation to populate the default info data.

# Deployment

There are two deployment options when using django, development where django’s own built in webserver serves the pages and production where a full webserver, such as Apache is used.

### General configuration

To install the required libraries a requirements.txt file is provided in the reference project. To install all of the libraries the following command can be run:

pip install –r requirements.txt

In both the development and production cases the database needs configuring before the first use. To do this change into the /reference/project directory and execute the following command:

./manage.py syncdb

An admin user should be created when prompted to be able to access the web based admin console.

## Development

The development version is a quick way of seeing the service in action and for development and testing. It can be run in this mode using the following command:

python manage.py runserver

This will launch the webserver running on port 8000 and allow it to be accessed at this base URL:

<http://127.0.0.1:8000/>

The celery server will also need running and can be started using the following command:

celery -q -A project worker --statedb=./celery.worker.state

## Production

When running in a production environment some change to the way the application is run need to be made.

### Celery

In production celery should be run as a daemon process, so if you use this technology in your own projects please see the following documentation for more details:

http://celery.readthedocs.org/en/latest/tutorials/daemonizing.html

### Django

In order to use a django application for anything more than development or testing then it needs to be configured to work with a webserver using the wsgi interface. The instructions below detail the steps that were necessary to run the application with the apache web server on a Centos 6 virtual machine running on CANARIE’s DAIR cloud. The steps start from the initial creation of the virtual machine and may not all be necessary depending on the deployment environment.

On a different system the specifics may change but the principle steps should remain similar.

## Create DAIR instance

This section deals with creating the DAIR instance and installing the correct dependencies. Initially create the DAIR instance using the following settings.

OS: CentOS

HD volumeVolume on /dev/vdc

Log in with an SSH session as per instructions in DAIR user docs

### Create reference user

sudo useradd reference

sudo passwd reference

### Add to sudoers

sudo visudo

Add line for reference user

reference All=(ALL) ALL

### Format and mount the volume as per the DAIR docs and change ownership to reference

### Install vim

sudo yum install vim

### Add paths to PATH environment variable

sudo vim ~/.bash\_profile

### Add these paths

/usr/local/bin

/home/reference/.local/bin/

### Turn off SE linux

sudo vim /etc/selinux/config

Change SELINUX=enforcing to SELINUX=permissive and save the file.

### Install apache

sudo yum install httpd

sudo yum install httpd-devel

Set apache to start on boot

sudo chkconfig --levels 235 httpd on

### Install development tools

sudo yum groupinstall "Development tools"

sudo yum install zlib-devel bzip2-devel openssl-devel ncurses-devel sqlite-devel readline-devel tk-devel

### In media/volume1 make tmp dir

cd /media/volume1

mkdir tmp

cd tmp

Build Python (VERY IMPORTANT TO USE ALTINSTALL)wget <http://python.org/ftp/python/2.7.3/Python-2.7.3.tar.bz2>

tar xf Python-2.7.3.tar.bz2

cd Python-2.7.3.tar.bz2

./configure --prefix=/usr/local --with-threads --enable-shared

make

sudo make altinstall

make clean

### Check default Python version still good

python –version

-> Python 2.6.6 (Good!)

### Check Python 2.7 dependencies

ldd /usr/local/bin/python2.7

linux-vdso.so.1 => (0x00007fffdb3ff000)

libpython2.7.so.1.0 => not found

libpthread.so.0 => /lib64/libpthread.so.0 (0x00007ff401945000)

libdl.so.2 => /lib64/libdl.so.2 (0x00007ff401741000)

libutil.so.1 => /lib64/libutil.so.1 (0x00007ff40153e000)

libm.so.6 => /lib64/libm.so.6 (0x00007ff4012b9000)

libc.so.6 => /lib64/libc.so.6 (0x00007ff400f25000)

/lib64/ld-linux-x86-64.so.2 (0x00007ff401b6c000)

The as seen above libpython2.7.so.1.0 library is not linked to the newly built python and so will need linking:

### Link libpython2.7.so.1.0

sudo ln -s /usr/local/bin/python2.7 /usr/bin/python2.7

sudo bash

echo "/usr/local/lib/python2.7" > /etc/ld.so.conf.d/python27.conf

echo "/usr/local/lib" >> /etc/ld.so.conf.d/python27.conf

ldconfig

exit

ldd /usr/local/bin/python2.7

…

libpython2.7.so.1.0 => /usr/local/lib/libpython2.7.so.1.0 (0x00007fa5f6721000)

…

pyhton2.7 –version

🡪 Python 2.7.3 (Good!)

### Install easyinstall, pip, virtualenv

wget <https://bitbucket.org/pypa/setuptools/raw/bootstrap/ez_setup.py>

sudo python2.7 ez\_setup.py

easy\_install-2.7 –version

-> setuptools 2.0

easy\_install-2.7 pip

easy\_install-2.7 virtualenv

### Build mod\_wsgi with Python2.7

wget <http://modwsgi.googlecode.com/files/mod_wsgi-3.4.tar.gz>

tar xvf mod\_wsgi-3.4.tar.gz

cd mod\_wsgi-3.4

./configure  --with-python=/usr/local/bin/python2.7

make

sudo make install

make clean

### Add mod\_wsgi to apache

sudo vim /etc/httpd/conf/httpd.conf

### Add to load module section

LoadModule wsgi\_module modules/mod\_wsgi.so

### Restart apache

sudo service httpd restart

### Configure iptables to allow http (Note: the number after INPUT is the position to insert and may vary)

sudo iptables -I INPUT 5 -p tcp -m state --state NEW -m tcp --dport 80 -j ACCEPT

sudo iptables -I INPUT 6 -p tcp -m state --state NEW -m tcp --dport 8080 -j ACCEPT

sudo iptables -I INPUT 7 -p tcp -m state --state NEW -m tcp --dport 443 -j ACCEPT

sudo iptables-save

# Generated by iptables-save v1.4.7 on Wed Dec 11 15:51:38 2013

\*filter

:INPUT ACCEPT [1:328]

:FORWARD ACCEPT [0:0]

:OUTPUT ACCEPT [4:736]

-A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT

-A INPUT -p icmp -j ACCEPT

-A INPUT -i lo -j ACCEPT

-A INPUT -p tcp -m state --state NEW -m tcp --dport 22 -j ACCEPT

-A INPUT -p tcp -m state --state NEW -m tcp --dport 80 -j ACCEPT

-A INPUT -p tcp -m state --state NEW -m tcp --dport 8080 -j ACCEPT

-A INPUT -p tcp -m state --state NEW -m tcp --dport 443 -j ACCEPT

-A FORWARD -j REJECT --reject-with icmp-host-prohibited

COMMIT

# Completed on Wed Dec 11 15:51:38 2013

sudo /sbin/service iptables save

sudo /sbin/service iptables restart

### At this point if we browse to the server using a web browser we should get a standard apache test page

http://<vm-ipaddress>

## Add django service code

### Make web dir

cd /media/volume1/

mkdir -p srv/www

### From external machine push trunk/rpi\_services

tar cvfz ref\_service.tar.gz reference

scp -i ~/.ssh/alberta\_rm\_reference.pem ref\_service.tar.gz centos@<vm-ipaddress>:

### Unpack tar (Back on server)

sudo cp /home/centos/ref\_service.tar.gz srv/www

cd srv/www

sudo chown reference:reference ref\_service.tar.gz

tar -xvf ref\_service.tar.gz

### Change ownership of dirs so apache can access

cd /media/volume1/srv/www

sudo chown -R reference:apache .

sudo chmod 775 reference/project

### Make virtual environment

cd /media/volume1

mkdir venv

cd venv

virtualenv --python=/usr/bin/python2.7 ENV

. ENV/bin/activate

python --version

Python 2.7.3 (Good! In ENV)

### Install python packages (ENSURE virtualenv still sourced, prompt starts with (ENV))

cd /media/volume1/srv/www/research\_software/reference

pip install –r requirements.txt

## Configure an apache Virtual Host to serve the django pages

sudo vim /etc/httpd/conf/httpd.conf

### At the bottom of the file uncomment

#NameVirtualHost \*:80

### Add the following (ServerName etc can be configured as appropriate to the system)

WSGISocketPrefix /media/volume1/run/wsgi

<VirtualHost \*:80>

ServerName www.example.com

ServerAlias example.com

ServerAdmin webmaster@dummy-host.example.com

WSGIDaemonProcess mygroup processes=2 threads=15 display-name=%{GROUP} python-path=/media/volume1/srv/www/reference/project:/media/volume1/srv/www/reference/project/project:/media/volume1/venv/ENV/lib/python2.7/site-packages

WSGIProcessGroup mygroup

WSGIScriptAlias / /media/volume1/srv/www/reference/project/project/wsgi-test.py

<Directory /media/volume1/srv/www/reference/project>

Order allow,deny

Allow from all

</Directory>

ErrorLog /media/volume1/srv/www/reference/project/logs/apache\_err.log

CustomLog /media/volume1/srv/www/reference/project/logs/apache\_access.log combined

</VirtualHost>

### Configure WSGISocketPrefix dir for wsgi

cd /media/volume1/

mkdir -p run/wsgi

sudo chown -R apache:apache run

### Restart apache

sudo service httpd restart

### At this point if we browse to the server using a web browser we should get a ‘Hello World’ test page from WSGI

http://<vm-ipaddress>

### Change Virtual Host to see correct django project:

sudo vim /etc/httpd/conf/httpd.conf

### Change

WSGIScriptAlias / /media/volume1/srv/www/reference/project/project/wsgi-test.py

### To

WSGIScriptAlias / /media/volume1/srv/www/reference/project/project/wsgi-prod.py

### Browse to application

http://<vm-ipaddress>/reference/service/app

1. https://www.python.org/ [↑](#footnote-ref-1)
2. https://www.djangoproject.com/ [↑](#footnote-ref-2)
3. http://www.django-rest-framework.org/ [↑](#footnote-ref-3)
4. http://www.celeryproject.org/ [↑](#footnote-ref-4)
5. http://www.python-requests.org/ [↑](#footnote-ref-5)