# Introduction

This application note describes the CANARIE Research Middleware Reference service. This service demonstrates a simple implementation of the NEP-RPI API as described in the document entitled “Research Service Support for the CANARIE Registry and Monitoring System”, revision 3. The aim of this document is explain the design of the service, how to use it and some notes on deploying it in a production environment.

The service is written in Python[[1]](#endnote-1) using the django[[2]](#endnote-2) web development framework and the django-rest-framework[[3]](#endnote-3).

The reference service functionality has deliberately been kept very simple.

# Service Design

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

reference

project

canarie\_api

project

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\_api directory. The main files used for the reference service here are:

models.py – Contains the data 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 service.

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.

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 reference/project/canarie\_api/urls.py file where is compared to the URLs configured there. If a match is made, the URL is passed to the appropriate handler in views.py.

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 application that the API is reporting on. This sample application has deliberately been kept a very simple in order to just outline the use of the API. 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 third section contains utility methods to support the API and application.

# How to use

This section will explain how to use the application.

To use the application browse to the main application page at:

<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>””

}

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

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

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

python manage.py syncdb

For this application it is not necessary to create an admin user.

### 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/

### Production

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/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))

pip install django

pip install djangorestframework

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