Installation of p3s and its software dependencies

Software dependencies

p3s consists of the server and client components. The code is organized in the folders **promptproc** and **client** respectively. There is an additional (and optional) service which displays the results of the p3s workflows if so desired. Folder **display** is a placeholder for this code which is in development.

The software dependencies of the server and the client are listed below. They obviously must be installed before the code can run.

p3s server dependencies

- Python3.5
- Django 1.10+
- django-tables2
- RDBMS (e.g. PostgreSQL; sqlite is used for development puprposes only and won't be suitable for deployment)
- psycopg2 (for PostgreSQL). A few minor dependendices will have to be resolved here.
- Apache 2.4 + mod_wsgi built for Python 3.5. This will likely require building Python from source with "enable_shared" option so that Python runtime is available for dynamic link to mod_wsgi.
- NetworkX 1.11 (1.10+ will probably do).

p3s client dependencies

- Python3.5
- Django 1.10+ (to include full timezone functionality)
- NetworkX 1.11

Known issues

- NetworkX: some versions other than 1.11 **may** present compatibility issues, resolving this is reponsibility of the developer
- Apache 2.4 deployment is different on different flavors of Linux, this must be addressed during deployment

- One must make sure that mod_wsgi is built for correct version of Python; this may be checked in a few ways, one method is to run “ldd” to look at the version shared libraries

Runtime environment

On distributed systems where the dependencies as listed above cannot be satisfied at the site level (e.g. installed by the administrators of the cluster on all worker nodes, servers etc) one can use Python virtualenv facility.

Installation and setting up

Getting the software

At the time of writing, the simplest way to get the code is by cloning the repo https://github.com/DUNE/p3s.git

Please DO NOT COMMIT/PUSH ANYTHING TO THIS REPO unless you contact the developer (M.Potekhin). Effort will be made to package the code for a more robust installation method.

Contents of the installation

p3s installation tree contains the following directories which are necessary for its function:

- **promptproc** - the core of p3s - the workflow managment and monitoring server
- **clients** - an assortment of clients for creation and manipulation of the objects residing in the server, mostly tailored to support a specific class of objects (job, pilot etc).
- **configuration** - a few very simple scripts created for convenience and used to set a few environment variables necessary for proper client operation and some aspects of the server operation. Default values will be assumed if the requisite environment variables are missing but proper operation is not guaranteed.

Other directories not critical for the system functionality:

- **inputs** contains files that can be used as examples and/or templates for testing and development of p3s suitable for production.
- **documents** (self-explanatory)

- **sandbox** - collection of arbitrary snippets of code kept as notes and not guaranteed to work at all; of interest only to the developers and won't be helpful for testers or users.
- **tools** - placeholder for general p3s tools not fitting into the "client" category

There is an additional project under the p3s umbrella, contained in the folder **display**. It's not a part of the core system and should be ignored until further notice.

DB Connection and other deployment-specific information

The database connection infoparameters are a part of "local configuration" since it may depend on deployment. This info is kept in the file named "local.py" which should exist in the same directory as manage.py. It is maintained outside of version control (i.e. is not in the repo) since it may contain sensitive information.

The template is as follows:

```
SITE = {
    'dirpath': '/eos/experiment/neutplatform/protodune/np04tier0/p3s',
    'p3s_input': '/eos/experiment/neutplatform/protodune/np04tier0/p3s/input',
    'p3s_output': '/eos/experiment/neutplatform/protodune/np04tier0/p3s/output',
    'dqm_domain': 'p3s-content.cern.ch',
    'dqm_host': 'p3s-content',
    'p3s_users': 'All,maxim,mxp,dladams',
    'p3s_services':'All,pcalc,T0,purge,tscan,olddata',
    'p3s_jobtypes':'purity,evdisp',
}

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'p3s',
        'USER': '********',
        'PASSWORD': '********',
        'HOST': '188.185.85.205',
        'PORT': '',
    }
}
```

Initializing the Database

The p3s server will utilize the RDBMS specified in its "settings" file. For example, it can run with the sqlite database which comes packaged with Django. However, production-grade applications will need a more powerful RDBMS such as PostgreSQL, MySQL, ORACLE etc, and one specific reason for that is proper implementation of the table and/or row lock for updates. This is not provided by sqlite.

If in development and utilizing the sqlite DB, it won't be initialized when you first get hold of the code, so this needs to be done as follows:

- change directory to p3s/promproc
- locate the file p3s/promptproc/promptproc/settings.py
- make sure it refers to sqlite
- run the following commands
- ./manage.py makemigrations
- ./manage.py migrate

Procedures will be similar for PostgreSQL but typically not encountered by ordinary users.

Define at least one data type

Meaningful testing requires that at least one datatype is defined in the system which should match whatever datatype(s) you use in your workflow template (DAG).

In the examples which come with the code the ".txt" extension is often used to demonstrate basic functionality. The system needs it to be able to autogenerate file names if necessary.

If your files all have extension '.txt' you may define the datatype "TXT" (which again must be consistent with you DAG) and use the following command to add the datatype to the server:

./dataset.py -R -j '{"name":"TXT", "ext":".txt", "comment":"testing text files"}'

If you no longer need the datatype you created (perhaps as the result of testing), it can be removed from the server as folllows:

./dataset.py -D test

The Environment

p3s clients will parse and utilize a variety of command-line arguments. It's often more convenient to make use of the environment variables such as the URL of the server, for consistency and to save on typing/scripting. The environment may be initialized by using scripts like ones in the "configuration" directory. One example is the environment variable "P3S_SERVER": it should have format similar to "http://myP3Sserver.cern.ch:8008/"

Logs

Logfiles are kept in the directories "/tmp/p3s" and "/tmp/username/p3s/*", in the latter case there are separate folders for pilots, workflows etc.

Pilots

p3s won't be able to do anything useful if there are no pilot jobs running at the available and properly configured computing resource. The p3s pilot job is implemented as a Python script kept in the "clients" directory.

In order to keep interfaces clean it was decided to leave up to the user (or rather the engineer tasked with deployment) how to deploy pilots rather than code specific solution into p3s itself. For example, this can be done even by running the pilot command interactively or via pdsh if so desired, and/or scripts can be created to interface the target batch system. One example of such implementation exists in the "dqmconfig" project which can be located on GitHub.

Running the development server

Using the development server should only be done in testing/educational scenarios. It is recommende that you use the single-threaded mode of operation and use the following command to start the server:

./manage.py runserver –nothreading

which again is done from the directory p3s/promprproc

This will default to the port 8000 and the server will be visible at the address localhost:8000 (plus maybe some bits of URLs depending on the version).

If you use ssh to connect the machine which runs the server, you could use ssh tunneling to get access to this host and port combination.

Embedded documentation

Many directories contain .md files that are easy to read in the browser on GitHub by just clicking on a file.

If you want to convert any of the .md files to PDF for better readability and a pretty printout, the following utility may be used which is easy to install on Linux - see this example:

pandoc -s -o README.pdf README.md

Learning about the clients

All clients have the ‘-h’ or the equivalent ‘–help’ option which summarizes the command line syntax, in addition some have the ‘usage’ option which may provide more info.