Table of Contents

PanDjango

About

Pandjango is an evolved version of PandaMonitor, built with the Django Web Framework and other tools to create an interactive and AJAX-capable client. See Appendix for more detail.

How-to

Installation

The code can be obtained in https://svn.cern.ch/reps/panda/panda-monitor/current/pandamonitor/pandjango.

Check it out of SVN into a directory of your choice (to which we shall refer as **pandjango_directory**). Make sure you have **Django 1.2** or later installed on your system, and **Python 2.4.3** or later. We will refer to the directory containing Django code as **django_directory**. Example:

```
ls /usatlas/u/mxp/Django-1.2.1/
AUTHORS   django   docs   extras   INSTALL   LICENSE   MANIFEST.in   PKG-INFO   README   scripts   setup.cf
```

You need to make sure that your **PYTHONPATH** contains the following components:

- pandjango_directory
- django_directory
- django_directory/django

Configuration

We enhanced the usual **settings.py** file that typically exists in a Django application, by adding useful configuration data which is Pandjango-specific. For example, particular views (functions) whose result we prefer to cache, are listed in a special dictionary:

```
CACHE_LIFETIME = {
    'servicelist':      3600*24,
    'sites':            3600*24,
    'pilotTypes':       3600*24,
    'scheds':            3600*24,
    'getErrors':        3600*24
    }
```

If you want to avoid caching results of a particular function, simply remove or comment out corresponding entry. The value in the dictionary specifies the lifetime of cached data, in seconds. It is recommended that you create a directory /tmp/cache, or any directory of your choice, and list in settings under **CACHE_DIR**. When the **CACHE_MODE** is 'SIMPLE', the application will put cache files (with names that are in fact url-encoded strings representing actual query URLs) in that directory. That makes it quite easy to navigate, flush and inspect during development. For anything closer to production the cache mode should be set to **DJANGO**, in which case you will also need to set up a caching location if file system is used, or start and maintain an instance of *memcached* if that is your choice.

If you want to bypass caching altogether, simply empty the dictionary **CACHE_LIFETIME**.

In addition, the application has the capability to hide whole sections of the web interface, if its needed for presentation or security reasons in beta testing. A section is a set of pages and widgets grouped under names like "autopilot", "production" etc. If you don't want to expose the section name "clouds", add this string to the collection **HIDE** in settings.py. No application code changes are necessary.

Database access

Your pandjango_directory will also need to contain file **dbaccess.py** (not in SVN), which has the following template and where you will need to fill in the credentials. Keep this file under chmod 400 to prevent unauthorized access:

```
def dbaccess():
    db = {
        'default':{'NAME':'INTR','ENGINE':'django.db.backends.oracle',   'USER':'***','PASSWORD':
        'atlas_panda':{'NAME':'ATLAS_PANDA','ENGINE':'django.db.backends.oracle','USER':'***','PA
        'atlas_pandamon': {'NAME':'ATLAS_PANDAMON','ENGINE':'django.db.backends.oracle','USER':'*
        'old':{'NAME':'INTR','ENGINE':'django.db.backends.oracle','USER':'***','PASSWORD':'***'},
        'prodsys':{'NAME':'atlas_prodsys','ENGINE':'django.db.backends.oracle','USER':'***','PASS
        }
    return db
```

Running the test server

You will first need to make sure you can start the development server that comes as part of Django installation. In the Pandjango directory, use the following command:

```
python manage.py runserver
```

By default, the server will start and open port 8000. If you want to change the port number, add it as additional argument to the command above. Now, start a browser of your choice on same machine as the server and point it to **http://localhost:8000/autopilot**. You should be directed to autopilot pages. Top of the page will contain a navigation bar, where currently only two sections have any content - *autopilot* and *production*.

A good way to debug the application is to look a the **JSON** messages it serves. For that, one should point the browser to the URL like ones found in "links" function in **misc.py**, and/or look in Javascript client code for further examples.

Running under Apache

Apache deployment is only recommended with a thoroughly tested instance of the code, with "DEBUG" option unset in the configuration file (settings.py) in order to not expose names of URLS and their mappings to functions, which is sensitive information. An excerpt from the requisite httpd configuration file is shown below:

```
Listen 20005
```

URLs

When running a test instance locally, the URL you would need to put into a browser is, as an example,

```
http://localhost:8000/autopilot
```

The "autopilot" part indicates that this is a part of hierarchy (a "section") of the monitor with functionality similar to "Autopilot" link in the original Monitor. Similarly, it can be "production" or "clouds".

When running under a proper server, same logic applies, of course the URL needs to point to the correct server address and port number.

Note on code organization

As we already mentioned in Configuration notes, the application can be thought of as consisting of "sections" which represent Panda monitor pages such as "Autopilot", "Clouds" etc. We chose to segregate corresponding server code into python code blocks under same names, e.g. **autopilot.py**, **production.py** etc. The matching parts of the client code is contained pandjango_directory/include/js/autopilot etc. That greatly facilitates team development.

Appendix (historical notes)

Motivations

There are a few factors that led us to consider an upgrade of Panda Monitoring system, which should be considered in conjunction with providing a feed to other Atlas systems such as a new dashboard etc. One is that the current Panda Monitor technology is becoming outdated:

- In current implementation, HTML “templates” are stored and versioned as in-line inclusions in the Python code (unwieldy and not conducive to presentation layer evolution and rapid development)
- Database access is done through explicit queries, thus the code is coupled to RDBMS
- Database access if synchronous, which sometimes means lack of responsiveness
- In a few instances, security aspects needed to be implemented in the code (e.g. avoiding SQL injection and such)
- Code base has become large and complicated enough, and difficult to maintain
- Extension to other systems (such as feeding data to a proposed new central Atlas monitoring system) are possible but can be cumbersome
- We need easier ways to create more specialized pages for VOs outside Atlas, as well as simpler and tailored interfaces for Atlas users

Solutions

What we see as solutions:

- decoupling data preparation from presentation
- AJAX-capable pages that can support fast, asynchronous page builds with successive levels of detail (whether prepared in a client-oriented way like GWT or a server-oriented way like current monitor code)
- a clean interface to external clients who want the data, such as the Dashboard
- better code structure, modularity and maintainability
- leveraging well supported, appropriate and standard tools/technologies/protocols

Choice of technology platform

We intend to use a Web Application Framework, and implement asynchronous data exchange between client and server (e.g. AJAX). We note that

- Frameworks promote code reuse
- The central part of the Monitor function is and will be database access (which is the core functionality in frameworks)
- Use of web templates is crucial for proper code organization and evolution (and frameworks implement that)
- Security mechanisms and session management are important, and modern frameworks include all of that, no need to “roll our own”

- Web Application Frameworks are perfectly suited not just for serving web pages, but any sort of data/objects, and therefore can efficiently interface with the central monitoring system
- Successful Web Application Frameworks invariably have vibrant user and developer communities, with plenty of sources of knowledge and support available – might as well leverage that!

Based on ATLAS experience and knowledge base, we chose Django as our framework.

Choice of network API

Message passing appears to be the optimal choice of network API for Panda Monitoring. In this approach, external clients and services send HTTP requests to Panda Monitoring system (implemented as a Web service) and get back a message containing serialized data, which can be parsed into Web pages in the browser, or parsed by an application for further processing and aggregation. A number of solutions for the message format, such as XML or JSON.

Status

The project is code-named **Pandjango**. There is a working Pandjango server prototype, as well as a few sophisticated pages based on **jQuery** and **jQuery-UI**.

Major updates:
-- MaximPotekhin - 11-Oct-2010

Responsible: MaximPotekhin

Never reviewed

This topic: PanDA > PanDjango
Topic revision: r2 - 2010-10-13 - MaximPotekhin