DEVELOPING WITH REPOZE.ZOPE2

Plumbing Zope 2 Into the WSGI Pipeline

Chris McDonough
Agendaless Consulting
November 2007

DEVELOPING WITH REPOZE.ZOPE2

Plumbing Zope 2 into the WSGI Pipeline

Chris McDonough
Agendaless Consulting
November 2007

History

Revision 1 - November 2007

Revision 2 - May 2008

Overview

Since early 2004, The Python web development community has been consolidating around a standard deployment model in the form of WSGI (Web Services Gateway Interface), a specification which describes how servers talk to Python web applications. A packaging standard referred to loosely as "eggs" has also become popular since that time. Repoze is an effort to bring Zope technologies to the larger Python web development community by breaking Zope up into pieces that fit into a WSGI deployment model and which can be installed using eggs as the basis for deployment. A side effect of this effort is that it allows current Zope users to make use of WSGI technologies for development purposes. The focus of this document is on repoze.zope2, a piece of software which allows you to run Zope 2 applications under WSGI and install Zope 2 using eggs.

Issues

Zope 2, the application server, has been around since late 1998. Because it's almost ten years old, and because over that ten years, it has strived to maintain full backwards compatibility with older releases, it has historically been slow in adopting newer Python features and specifications. However, it is still very popular and in use by a large number of people. Plone, the content management system, is based on Zope 2, and continues to drive its adoption even today.

Zope 3, the application server, has been around since 2001. It is a rewrite of Zope from scratch. Its development was begun largely because it was thought that some design deficiencies in Zope 2 could only be fixed through a complete rewrite. No Zope 2 applications will run under Zope 3, although Zope 2 has been retooled continually over time to take advantage of some Zope 3 features by using Zope 3 packages as libraries. Zope 3 is composed of many libraries that can be used independently of their roles within a Zope 3 application server deployment. However, because most of these libraries make use of a Zope-specific "component architecture" that is not widely used by the larger Python community, and because their packaging focus is relative to their use within an application server deployment, using them is often difficult for people who would rather not understand the component architecture and the library's role within a Zope application server deployment.

Repoze attempts to make it easier to reuse bits of Zope 2 and Zope 3 by pulling select pieces of from the application server proper and turning each into WSGI "middleware" that can be used independent of any particular web server or application. Middleware is code with a well understood interface which sits between the WSGI server and the application endpoint. Middleware can influence exception behavior, perform body transforms, and do other useful things. A catalog of existing middleware (Repoze is not represented there, however) is present at http://www.wsgi.org/wsgi/Middleware_and_Utilities.

Repoze.zope2 is the installable piece of software that allows you to use Zope 2 applications largely unchanged within a WSGI environment. repoze.zope2 relies heavily on Ian Bicking's Python Paste, aka "Paste", which is a set of configuration mechanisms and generic WSGI middleware items that make it easier to deploy WSGI applications.

What's Better

Developing Zope 2 applications using repoze.zope2 is meant to be exactly like developing Zope 2 applications under a "classic" deployment. You should be able to use all the knowledge you've gained writing "plain" Zope 2 applications to write applications using repoze.zope2 but you may also take advantage of the benefits of WSGI. Several key benefits are:

1. Several previously hardcoded bits are now configurable in various ways because they've been moved out to middleware. For example: you can influence the ConflictError "retry" policy by changing Paste configuration. In Zope 2 the number of ConflictError retries that must occur for a request to fail is hardcoded to 3. You can use a much-simplified mechanism for doing virtual hosting by changing configuration. You can turn off transactions if you choose (e.g. for a "read-only" site, they are just overhead) by changing Paste configuration. Note that the simplest form of configurability is removal of these middleware pieces; Zope 2 will operate without any of them.

2. You can use different WSGI servers (including Apache via mod_wsgi) to serve up your Zope application by changing configuration.

3. You can employ various pieces of WSGI middleware to show error tracebacks in different ways in your web browser (e.g. egg:Paste#cgitb or egg:Paste#evalerror) while you develop.
4. You can use WSGI middleware to perform access and error logging (replacing the analogue of those features built in to Zope 2).
5. You can run non-Zope applications (e.g. Pylons, Roundup, static file serving, etc) in the same process space as your Zope application(s) by changing your Paste configuration.
6. You can skin dissimilar applications using applications like Deliverance, which is middleware that performs output transformation.
7. You can externalize some functionality (like authentication) using middleware.
8. You can create your own middleware pieces to perform deployment-specific tasks when it's more appropriate to do this than it is to understand some specific Zope internals (e.g. transform all HTML output, adding a footer).
9. Your software will be compatible with yet-to-be-created WSGI servers, which may offer specific benefits for various types of deployments.
10. You can use Repoze middleware with non-Repoze WSGI applications, which can provide a bit of development continuity for Zope developers when they need to develop non-Zope applications.

What's Different

1. The Zope 2 standard_error_message machinery does not currently work under repoze.zope2. This is because the Zope 'zpublisher_exception_hook' callback is not invoked when an error occurs. Instead, the httpexceptions middleware is responsible for displaying NotFound and Internal Server Error messages to the browser. In general, all exceptions, save for Zope's Redirect, NotFound and Unauthorized are propagated to enclosing middleware in the pipeline. More explicitly, the Redirect exception is turned into a paste.httpexceptions.HTTPFound error, which is later caught by the httpexceptions middleware and turned into an HTTP redirect. The NotFound exception is turned into a paste.httpexceptions.HTTPNotFound error, which is later caught by the httpexceptions middleware and turned into a 404 error. Unauthorized is caught internally so frameworks like PAS can handle them as necessary. It is expected that we'll make the standard_error_message machinery work in later releases of repoze.zope2.

2. The Zope 2 through-the-web "error_log" object will not show any exception history. Instead, you must use the repoze.errorlog '/__error_log__' URL to see an analogue of this information (or just view console output). This is also because the Zope 'zpublisher_exception_hook' callback is not invoked when an error occurs when running Zope 2 under Repoze.
3. The 'zopectl' process controller does not exist in a repoze deployment. Single-purpose scripts handle the functionality it used to provide. To start Zope as a daemon ("background mode"), run 'paster' under a supervisor process controller or use the --daemon option to 'paster serve'.
4. A pid file is not created in the sandbox 'var' directory, because Zope may not be the only application running under WSGI within a single process. A lock file is not created either, because it's only useful when Zope is run under 'zopectl'; repoze doesn't support it.
5. The 'effective-user' option in zope.conf has no effect for the same reasons.

Installing repoze.zope2 using **zc.buildout**

To install repoze.zope2 via **zc.buildout**, you should have an internet-connected UNIX system with Python 2.4 with setuptools installed and the capability to compile C code. Relative file paths referred to from here within this document should be considered relative to the "sandbox" you create.

Check out one of the buildouts from http://svn.repoze.org/buildouts. For example:

```
$ svn co http://svn.repoze.org/buildouts \
     http://svn.repoze.org/buildouts/repoze.zope2/trunk/ sandbox
```

Then execute the bootstrap command within the sandbox and run the buildout:

```
$ cd sandbox
$ python2.4 bootstrap.py
$ bin/buildout
```

Follow the instructions from there on in printed to your console.

Installing repoze.zope2 using Virtualenv

To install repoze.zope2 via virtualenv, you should have an internet-connected UNIX system with Python 2.4 with setuptools installed and the capability to compile C code. Relative file paths referred to from here within this document should be considered relative to the "sandbox" you create.

I. Create a virtualenv to House Your repoze.zope2 Sandbox

The repoze.zope2 installation routine depends on setuptools (see http://peak.telecommunity.com/DevCenter/setuptools). You'll need to install setuptools in your Python environment before you are able to install any Repoze packages. Install setuptools by downloading and running http://peak.telecommunity.com/dist/ez_setup.py (ez_setup.py) using the a Python 2.4 interpreter that you'd like to use with repoze.zope2. **Please note that repoze.zope2 doesn't work properly under Python 2.5. It requires Python 2.4.**

To to be able to create a repoze.{plone,plone,zope2} "sandbox" (an isolated Python environment), install the virtualenv package from http://pypi.python.org/pypi/virtualenv" package using the setuptools 'easy_install' command. You may need to be a root user to do this if your Python is the "system" Python

```
$ easy_install virtualenv
```

Virtualenv provides a way to create a "virtual" Python installation, which makes it easy to evaluate and deploy package without potential conflict from libraries you've already installed into your main Python's sys.path. Since repoze.zope2 and its dependencies require lots of packages, virtualenv is a good way to get a clean environment for your project. The virtualenv you create will also serve "double duty" as a repoze.zope2 "instance home".

As a *non-root-user*, create a virtualenv to house your sandbox (use any path other than '/path/to/env' that suits you):

```
$ virtualenv —no-site-packages /path/to/env
```

The '--no-site-packages' options ensures that the new Python virtual environment does not inherit any packages from the base Python installation.

II. Post-Virtualenv-Creation Steps

The following commands create a 'repoze.zope2' sandbox environment, including Zope 2 and all of its dependencies

```
$ cd /path/to/env
$ bin/easy_install -i http://dist.repoze.org/zope2/latest/simple \
      repoze.zope2
```

Once the 'repoze.zope2' package and its dependencies are installed, you must create 'instance' files using the mkzope2instance script installed into the virtualenv::

```
$ bin/mkzope2instance .
```

Then you can then run the derived server by executing

```
$ bin/addzope2user admin admin # create manager account
$ bin/paster serve etc/zope2.ini
```

Your setup is complete. You can log on to Zope at

http://localhost:8080/manage .

A Tour of the Default repoze.zope2 Sandbox

The directory created when you invoked ‘mkzope2instance’ is what we’re calling a “sandbox”. By default, inside that directory are the following directories:

Products — A directory in which to unpack classic Zope 2 Products for installation into the environment.

bin — the place where all scripts are installed, including scripts related to Repoze, Paste, Zope, virtualenv, ZODB, ZEO, and Python.

etc — the place where all configuration files are placed, including files related to Paste (zope2.ini), Zope (zope.conf and site.zcml), and mod_wsgi (apache2.conf).

import — a place where Zope .zexp files can be placed for import.

include — Python header files.

lib — a location for Python packages.

var — the place where Zope data files (e.g. Data.fs) go.

A Tour of Repoze Scripts in The Sandbox bin Directory

The sandbox “bin” directory contains a number of scripts that are related to Repoze.

addzope2user — add a management user to the root Zope acl_users user folder. Equivalent to “zopectl adduser”.

debugzope2 — start a Python interpreter with the root Zope object bound to “app”. Equivalent to “zopectl debug”.

```
runzope2script — run a Python script with the root Zope object bound to "app"
in the global namespace.  Equivalent to "zopectl run".
```

A Tour of the repoze.zope2 Paste Configuration

The default repoze.zope2 sandbox Paste configuration file (etc/zope2.ini) looks something like this:

```
[DEFAULT]
debug = True

[app:zope2]
paste.app_factory = repoze.obob.publisher:make_obob
repoze.obob.get_root = repoze.zope2.z2bob:get_root
repoze.obob.initializer = repoze.zope2.z2bob:initialize
repoze.obob.helper_factory = repoze.zope2.z2bob:Zope2ObobHelper
zope.conf = %(here)s/zope.conf

[pipeline:main]
pipeline = egg:Paste#cgitb
           egg:Paste#httpexceptions
           egg:repoze.retry#retry
           egg:repoze.tm#tm
           egg:repoze.vhm#vhm_xheaders
           egg:repoze.errorlog#errorlog
           zope2

[server:main]
use = egg:repoze.zope2#zserver
host = 127.0.0.1
port = 8080
```

The [DEFAULT] section contains a single key, "debug", which is set to true by default. This is a configuration flag that is respected by most Paste middleware which influences its behavior (akin to "debug mode" in Zope 2). Its influence on the behavior of the middleware is defined by the middleware itself. The [DEFAULT] section may contain other global settings, but for the purposes of repoze.zope2 it doesn't need to.

The [app:zope2] section is a section which defines the "zope2" WSGI application. The 'paste.app_factory' key is a Paste-specific key which points at a callable which returns a WSGI application. The other keys in the section are configuration related to "obob", which is an object publishing framework developed specifically for Repoze that allows us to publish both "straight" Zope 2 applications and applications that are very "Zope-y" but which are not actually Zope (e.g. repoze.kiss). *The existence of obob is a Repoze implementation detail. It's not required that you know anything about obob to use Repoze to develop and deploy Zope applications. You should not change the specific keys in the [app.zope2] section related to obob if you want your Zope to run under repoze.zope2.* The other key in the section is 'zope.conf', which simply points to the "normal" zope.conf file used by a Zope 2 deployment. The use of '%(here)s' in its value is an indirection. '%(here)s' is a Paste convention; it refers to the directory in which the Paste configuration file resides. In this case, the zope.conf file is in the same directory as the zope2.ini file.

The '[pipeline]' section establishes a Paste "pipeline". A pipeline consists zero or more pieces of middleware and, at the end, an application. In this configuration, the endpoint application (the "last" item in the pipeline) is "zope2", which is a reference to the application we defined within the '[app:zope2]' section. The middleware which precedes this application comes before this reference. Both middleware and applications may be referred to by a reference to another section in the configuration file (e.g. 'zope2'), by an 'egg name' (e.g. 'egg:repoze.errorlog#errorlog') or by a dotted path to a Python callable. The middleware which happens to compose the default pipeline for repoze.zope2 is as follows (in "reverse" order, before the 'zope2' endpoint):

- egg:repoze.errorlog#errorlog is a piece of middleware which catches exceptions and logs them to a Python logging channel. It also keeps a history of exceptions around that can be viewed by visiting the '/__error_log__' URL of the server (much like Zope 2's error_log object). This middleware is in the pipeline because Zope 2's error_log object does not function under repoze. It can also be used outside the context of Zope entirely.

- egg:repoze.vhm#vhm_xheaders is a piece of middleware which, if passed specially crafted variables in the WSGI environment, will perform the same duties as Zope's Virtual Host Monster: it adjusts CGI variables to include some intermediate path or change the hostname, or the port, etc, and allows you to specify the root at which the traversal should start. The spellings are different than Zope's VHM (mainly for simplicity), but the outcome is the same. See http://svn.repoze.org/repoze.vhm/trunk/README.txt for a description of the semantics and syntax. Again, this middleware can be used independent of Zope entirely.

- egg:repoze.tm#tm is a piece of middleware that implements a ZODB transaction management policy. When this middleware is present, when a WSGI request is started, a transaction begins. If the request finishes without an exception, the transaction commits. If the request encounters an exception, the transaction aborts. Although this transaction behavior is associated with ZODB (because the transaction package upon which it depends is not separable from the ZODB package, for silly reasons), it really has nothing to do with the "object database" part of ZODB. It can be used to begin, commit, and abort relational database transactions or even filesystem operations (e.g. blobs). This middleware is completely independent of Zope, save for its dependency on ZODB.

- egg:repoze.retry#retry is a piece of middleware that implements a "retry policy" for requests that raise certain exceptions. Its first and foremost duty is to retry requests ZODB "ConflictErrors" (ZODB uses optimistic concurrency), but the types of errors which it is willing to retry for is configurable, and you can configure the number of times that the request is retried. This behavior used to be part of the Zope publishing machinery, and was hardcoded to retrying conflict errors and hardcoded to retry at most three times. With this functionality as middleware, the exception type is changeable, and the retry count is changeable, and it can be used independently from Zope for any purpose.

- egg:Paste#httpexceptions is a piece of middleware which catches certain exception types (e.g. in the case of repoze.zope2, paste.httpexceptions.NotFound, paste.httpexceptions.HTTPFound, and paste.httpexceptions.Unauthorized) and modifies the environment, turning the response code into something that makes sense for those types of errors (e.g. 404 Not Found 302 Redirect and 401 Unauthorized, respectively).

- egg:Paste#cgitb is a piece of middleware which serves as a top-level exception handler, and displays the traceback nicely colored to the requesting browser. It is valuable to use during development, but it can be removed under production. You can also use fancier top-level exception handlers like Ian's "evalerror" (http://pythonpaste.org/screencasts/evalerror-screencast.html), which is very useful for debugging purposes.

The '[server:main]' section defines what type of HTTP server to use and its configuration parameters. The default repoze.zope2 configuration uses the so-called "ZServer" WSGI server from Zope 3 as its server (egg:repoze.zope2#zserver). It will listen on 127.0.0.1:8080 by default. Other servers that you can use include Twisted.Web2 (not via Paste), egg:Paste#http, egg:PasteScript#wsgiutils, and egg:Paste:cherrypy. For an overview of how to configure each, see http://blog.repoze.org/fitting_room-20071029.html.

Starting Zope2 Under Repoze

Immediately after installing repoze.zope2, you will need to create an administrative user so you can log into the Zope Management Interface. To do so, use the "addzope2user" command:

```
$ bin/addzope2user admin admin
```

This will add a user named "admin" with the password "admin."

To start your Zope server under the default configuration, use the “paster serve” command, like so:

```
$ bin/paster serve etc/zope2.ini
```

Printed to your console you will see output like this:

```
Starting server in PID 7887.
zserver on port 8080
```

Once you’ve started Zope, you can log in to the Zope Management Interface by visiting “http://localhost:8080/manage” in your browser. You will see the standard Zope Management Interface.

Products, Configuration Files, Log Files, and Data Files

Like a normal Zope installation, you can unpack Zope Products into the “Products” directory inside the sandbox. When you restart ‘paster’, the product(s) will be installed.

Unlike a normal Zope installation, you cannot unpack “normal” Python packages into the “lib/python” directory in the sandbox, as this location is not on the PYTHONPATH. Instead, place them into the “lib/python2.4/site-packages” directory. You may use the ‘bin/installproduct’ script to install Zope products as well.

You can edit the standard ‘zope.conf’ file by visiting it within the ‘etc’ directory of the sandbox. All options, save for the ones related to servers (because you’re no longer using the built-in Zope ZServer), work as you would expect. The default setup puts ZODB data into a FileStorage that is kept in ‘var/Data.fs’. You can set up a ZEO server and use it “as normal” by running the “mkzeoinst” command in the sandbox’s “bin” dir and progressing through its questions normally. When you’re done, change the <zodb_db main> stanza in the ‘zope.conf’ file to a ClientStorage.

By default, Zope is configured to log its event log to stdout instead of any particular log file. You can change this by editing zope.conf’s “logger event” section.

Debugging

Although the venerable “zopectl debug” no longer exists under repoze.zope2, an equivalent command “debugzope2” exists in the bin directory. When you invoke it, you will see a familiar debugging stanza:

```
Python 2.4.4 (#1, Oct 17 2007, 20:25:32)
[GCC 4.0.1 (Apple Computer, Inc. build 5250)] on darwin
```

```
Type "help" for more information. "app" is the Zope application object.
>>> app
<Application at >
>>>
```

Tricks

For fun and profit, replace in the Paste pipeline 'egg:Paste#cgitb' with 'egg:Paste:evalerror'. Evalerror is a useful and fun piece of middleware that allows you to inspect the frame stack of a particular request that raises an exception through the browser used to invoke the request.

Try out 'repoze.tempita' (see repoze.org) as an example of output transformation middleware. Every request with '${dollar-braced}' format strings in its output will allow for textual replacement using static definitions within the Paste configuration file.

Try using repoze.zope2 under mod_wsgi in Apache. mod_wsgi is an Apache module which allows you to serve WSGI applications from within Apache's process space. A sample configuration stanza for mod_wsgi exists at
http://svn.repoze.org/repoze.zope2/trunk/repoze/zope2/etc/sample-apache2.conf .

Try removing the 'egg:repoze.tm#tm' middleware from the Paste configuration. Notice that requests to Zope function properly but no changes made by requests are actually committed.

repoze.plone

"repoze.plone" is an installable egg that depends on repoze.zope2, and includes all the software reqired to run Plone under repoze.zope2. You can install it by using the steps to install repoze.zope2 discussed in a previous section, but use the index at
http://dist.repoze.org/plone/latest/simple, and use the eggname "repoze.plone". When you're finished installing it (it's the same as installing repoze.zope2), log in to the ZMI and add a "Plone Site".