stargate Documentation

Release 0.5

Ben Ford

March 04, 2013

CONTENTS

stargate is a package for adding [WebSockets] support to pyramid applications using the excellent eventlet library to handle long running connections inside a non blocking WSGI server.

CHAPTER

ONE

ADVANTAGES

Most existing implementations of a WebSocket capable server run stand alone from your web app, usually on a different port or host. `stargate` allows you to connect persistent connection directly to the same objects that comprise your application. The advantages of this approach will become apparent in the examples section.

Contents:

1.1 What are WebSockets?

From wikipedia:

```
WebSockets is a technology providing for bi-directional, full-duplex
communications channels, over a single TCP socket
```

The WebSocket protocol provides a persistent low latency, low complexity way to achieve two way communication from the browser to server.

From a client point of view using WebSocket is very simple:

```
var ws = new WebSocket('ws://somehost/some/url');
ws.onopen = function(msg){
    //do some cool setup
}
ws.onmessage = function(msg){
    //do something cool
}
ws.onclose = function(msg){
    //do some clean up... stay cool
}

// later:
ws.send('How cool am I?!');
```

That's pretty much all there is to it.

1.1.1 How do they work?

The socket connection that WebSocket uses is negotiated with an HTTP request/response between the client and the server

On the server side things are slightly more complex. The server must:

- Perform a handshake with the client

- Keep hold of the persistent connection
- Receive messages from the client
- Route messages to the client

The following refers to the hybi version 10 websocket spec [hybi10]

Handshake

From the browser:

```
GET /ws HTTP/1.1
Host: pmx
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Version: 6
Sec-WebSocket-Origin: http://pmx
Sec-WebSocket-Extensions: deflate-stream
Sec-WebSocket-Key: x3JJHMbDL1EzLkh9GBhXDw==
```

The server sends back:

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: HSmrc0sMlYUkAGmm5OPpG2HaGWk=
```

Note: There are also mechanisms for sub-protocols and extension. These haven't been implemented yet

---

See the [WebSockets] Wikipedia page for more details.

The part of the protocol that deals with framing is much more extensive than the older versions. Framing is handled by the ws4py library.

1.1.2 Older Versions

Note: The following is included for historical reasons only - as I understand it these have been disabled by default for security reasons in most browsers.

Handshake

The handshake version of the WebSocket protocol underwent a major revision at version 76 [ws76]. As of version 0.2, Stargate supports both this and the older [ws75]. Examples of both are below:

Version 76

From the browser:

```
GET /demo HTTP/1.1
Host: example.com
Connection: Upgrade
```

```
Sec-WebSocket-Key2: 12998 5 Y3 1  .P00
Sec-WebSocket-Protocol: sample
Upgrade: WebSocket
Sec-WebSocket-Key1: 4 @1  46546xW%0l 1 5
Origin: http://example.com

^n:ds[4U
```

The server would send back:

```
HTTP/1.1 101 WebSocket Protocol Handshake
Upgrade: WebSocket
Connection: Upgrade
Sec-WebSocket-Origin: http://example.com
Sec-WebSocket-Location: ws://example.com/demo
Sec-WebSocket-Protocol: sample

8jKS'y:G*Co,Wxa-
```

Version 75 and older

Client:

```
GET /demo HTTP/1.1
Host: example.com
Connection: Upgrade
Upgrade: WebSocket
Origin: http://example.com
```

Server:

```
HTTP/1.1 101 WebSocket Protocol Handshake
Upgrade: WebSocket
Connection: Upgrade
WebSocket-Origin: http://example.com
WebSocket-Location: ws://example.com/demo
```

The implementation of the handshake can be found in `stargate.handshake`

Connection

Once the handshake has been successfully negotiated there is a persistent bi-directional websocket connection from the client to the server. This is a pretty thin wrapper around a socket that sends text messages packed in the `\x00` and `\xFF` bytes.

1.2 Examples

The following is a simple one page example of using `WebSocketAwareResource` and `WebSocketView` with [Traversal] to control persistent objects in the `resource tree`. These persistent objects communicate the control changes to connected clients via a [WebSockets] connection.

```
1 import eventlet
2 from eventlet import wsgi
3 from paste.httpserver import serve
```

```
from pyramid.config import Configurator
from pyramid.response import Response
from pyramid.view import view_config
from stargate import WebSocketAwareResource, WebSocketView, is_websocket
import simplejson as json

host = "127.0.0.1"
port = 9999

home_html = """\
<html>
    <head>
        <script type="text/javascript" src="http://ajax.googleapis.com/ajax/libs/jquery/1.6.4/jquery
        <script>
            $(function() {
    ws = new WebSocket("ws://%(host)s:%(port)s/jobs/1/");
    ws.onmessage = function(msg) {
        $("body").append("<p>" + msg.data + "</p>");
    };
    STARTED = false;
    $("#start-stop").click(function(evt) {
        $.post("/jobs/1/", {state: STARTED ? "stop" : "start"}, function(result) {
            STARTED = !STARTED;
            $('#start-stop span').text(STARTED ? "on" : "off");

        });
    });
});
        </script>
    </head>
    <body>
        <h1>Hi</h1>
        <button id="start-stop">Job 1 <span>off</span></button>
    </body>
</html>
""" % dict(host=host, port=port)

class JobRoot(object):
    """A container for jobs

    Gets or creates Job objects for certain ids
    """

    def __init__(self):
        self._jobs = {}

    def __getitem__(self, item):
        try:
            return self._jobs[item]
        except KeyError:
            return self.create_job(item)

    def create_job(self, id):
        job = Job(id, self)
        self._jobs[id] = job
        return job


```

```
62  class Job(WebSocketAwareResource):
63      """This is a permanent object.
64
65      It's responsible for maintaning a list of connected clients (websockets)
66      and updating them when its state changes
67      """
68
69      def __init__(self, id, parent):
70          self.__name__ = id
71          self.__parent__ = parent
72          self.state = "OFF"
73
74      def control(self, state):
75          """This function updates the state
76
77          Its called by the control function (in response to a post)
78          It triggers the sending of self.state to all connected clients. If you
79          connect multiple browsers (or tabs) they will all be updated
80          """
81          self.state = state
82          self.send(state)
83
84  class JobView(WebSocketView):
85      """The view connects pyramid with the resource
86
87      In this simple example it simply adds the websocket to the Job's listeners.
88      It then goes to sleep *blocking the thread it's in* this is where eventlet
89      comes in. In real life you'd do things like listening for updates and
90      handling messages coming in on the websocket in the while block.
91      """
92
93      def handler(self, websocket):
94          job = self.request.context
95          job.add_listener(websocket)
96          while True:
97              eventlet.sleep(60)
98
99  def control(job, request):
100     """Post to this view to set the state
101
102     this will trigger Job to report the state to connected clients
103     """
104     state = request.POST.get("state")
105     if state:
106         job.control(state)
107     return dict(id=job.__name__, state=job.state)
108
109 class Root(dict):
110     """The root of url traversal"""
111
112     def __init__(self):
113         super(Root, self).__init__(jobs=JobRoot())
114
115 def home(request):
116     """Serves up home_html, setting up a simple js demo"""
117     return Response(home_html)
118
119 root = Root()
```

```
120
121 def root_factory(request):
122     return root
123
124 if __name__ == '__main__':
125     config = Configurator(root_factory=root_factory)
126     config.add_view(home, context=Root)
127     config.add_view(JobView, context=Job, custom_predicates=[is_websocket])
128     config.add_view(control, context=Job, renderer="json", xhr=True)
129     app = config.make_wsgi_app()
130     listener = eventlet.listen((host, port))
131     wsgi.server(listener, app)
```

1.3 Modules

1.3.1 stargate

WebSocket support for pyramid is implemented as a `view` which handles the upgrade request and a `resource` which manages the persistent connected clients. Both should be subclassed to provide send and receive functionality desired

`stargate.`**`is_websocket`**(*context*, *request*)
Custom predicate to denote a websocket handshake

See [predicate_arguments] and the `custom_predicate` key word argument to `repoze.bfg.configuration.Configurator.add_view()`

`WebSocketView` and `WebSocketAwareResource` are also available in this namespace, and this is the prefered location to use them from

1.3.2 stargate.resource

This module supplies a class which can be subclassed or mixed in and used in conjunction with `WebSocketView`. It provides functionality for adding and removing `websockets` as listeners for events

class `stargate.resource.`**`WebSocketAwareResource`**
An object in a *pyramid:traversal* graph that handles websockets

It is designed to be persistent and to route messages to attached clients

`add_listener`(*ws*)
Adds a `eventlet.websocket.WebSocket` the the set of listeners

`listeners` = **None**
A set of attached `websockets`

`remove_listener`(*ws*)
Removes ws from the set of listeners

`send`(*message*)
Sends `message` to all sockets in the set of `listeners`

It will clear up any websockets that are no longer connected

1.3.3 stargate.view

exception stargate.view.**IncorrectlyConfigured**
Exception to use in place of an assertion error

class stargate.view.**WebSocketView**(*request*)
A view for handling websockets

This view handles both the upgrade request and the ongoing socket communiction.

handle_upgrade()
Completes the upgrade request sent by the browser

Sends the headers required to set up to websocket connection back to the browser and then hands off to handle_websocket().

See [websocket_protocol]

Returns webob.exc.HTTPBadRequest if handshake fails

handle_websocket(*websocket*)
Handles the connection after setup and handshake is done

Hands off to handler() until the socket is closed and then ensures a correct webob.Response is returned after the socket is closed

Parameters **websocket** – A WebSocket

handler(*websocket*)
Handles the interaction with the websocket after being set up

This is the method to override in subclasses to receive and send messages over the websocket connection

Parameters **websocket** – A WebSocket

1.3.4 stargate.factory

This module provides a paste [server_factory] to run pyramid inside an eventlet wsgi server

See Paste Deploy for more details.

stargate.factory.**server_factory**(*global_conf*, *host*, *port*)
Implements the [server_factory] api to provide an eventlet wsgi server

1.3.5 stargate.handshake

The handshake module contains implementations of different websocket spec versions' handshakes.

The WebSocket spec had a major revision at version 76 [ws76] on May 6th 2010. This module is an attempt to insulate downstream application programmers from those changes

exception stargate.handshake.**HandShakeFailed**
Raised when the handshake fails

exception stargate.handshake.**InvalidOrigin**
Raised when the websocket request doesn't have a valid origin

stargate.handshake.**build_location_url**(*headers*)
Construct a websocket url for given headers

Parameters **headers** – webob.headers.EnvironHeaders

stargate.handshake.**handshake_pre76**(*headers*, *base_response*)
The websocket handshake as described in version 75 of the spec [ws75]

Parameters

- **headers** – The request headers from `websocket_handshake()`
- **base_response** – The headers common across different spec versions

stargate.handshake.**handshake_v76**(*headers*, *base_response*)
The websocket handshake as described in version 76 of the spec [ws76]

Parameters

- **headers** – The request headers from `websocket_handshake()`
- **base_response** – The headers common across different spec versions

stargate.handshake.**websocket_handshake**(*headers*, *allowed_origins=None*)
Perform the websocket handshake

This function does the part of the handshake that is common across spec versions and then hands off to the spec specific implementations.

See: `handshake_pre76()`

Parameters **headers** (`webob.headers.EnvironHeaders`) – The websocket upgrade request headers (headers attribute from `webob.Request`)

Raises `HandShakeFailed`, `InvalidOrigin`

Returns A string to send back to the client

References

1.4 Changelog

1.4.1 0.4

- Add support for the HyBi version of WebSockets (specifically version-10). This version was released to solve some previous security concerns and

will be the version in Firefox 7/8 and Chrome 14 onwards accoring to Wikipedia. - Introduces a dependency on ws4py

- See http://tools.ietf.org/html/draft-ietf-hybi-thewebsocketprotocol-10

1.4.2 0.3

- Fix bug in setup.py
- Improve documentation and consistency with pyramid naming

1.4.3 0.2

- Support for version 76 of the websocket handshake

1.4.4 0.1

- Initial release, ported from rpz.websocket and repoze.bfg to pyramid

CHAPTER

TWO

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

BIBLIOGRAPHY

[WebSockets] http://en.wikipedia.org/wiki/Web_Sockets

[hybi10] http://tools.ietf.org/html/draft-ietf-hybi-thewebsocketprotocol-10

[ws76] http://tools.ietf.org/html/draft-hixie-thewebsocketprotocol-76

[ws75] http://tools.ietf.org/html/draft-hixie-thewebsocketprotocol-75

[Traversal] https://pylonsproject.org/projects/pyramid/dev/narr/traversal.html

[server$_{factory}$] http://pythonpaste.org/deploy/#paste-server-factory

[websocket$_{protocol}$] http://en.wikipedia.org/wiki/Web_Sockets#WebSocket_Protocol_Handshake

[predicate$_{arguments}$] http://docs.repoze.org/bfg/1.3/narr/views.html#predicate-arguments

PYTHON MODULE INDEX

S