Zerogw Documentation

Release 0.5.9

Paul Colomiets

February 12, 2015

Contents

Contents:

CHAPTER 1

Installation Guide

1.1 Prerequisites

- libzmq >= 2.1
- openssl
- libev
- libyaml

1.2 Easy Way

We currently have only packages for Archlinux and Ubuntu.

1.2.1 Archlinux

yaourt -S zerogw

1.2.2 Ubuntu

apt-add-repository ppa:tailhook/zerogw apt-get install zerogw

1.3 Hard Way

To compile from source or to compile recent version from git you need the following dependencies:

- python3 needed for coyaml to build configuration parser
- libwebsite for handling http
- coyaml for handling configuration
- libzmq and libev of course
- libyaml for parsing configuration

For compiling coyaml, libwebsite and then zerogw itself you need to run the following magick sequence of commands for each of them:

```
./waf configure --prefix=/usr
./waf build
sudo ./waf install
```

For other tools see respective documentation.

1.4 Verifying Install

After install is completed you can run:

```
zerogw
```

It should print few warnings on screen. Startup messages are printed as warnings since they are important for installations where lower level messages disabled. Also usually your mime-types is inconsistent (at least in archlinux it is). But nevermind, just got to the browser and check `http://localhost:8000/hello`, you should see some greeting if everything works.

CHAPTER 2

Zerogw Backend Protocols

2.1 HTTP Forwarding

We use zeromq's request reply model for http forwarding (except for *long polling case*)

2.1.1 Request

For each http request zerogw forwards a single multipart message. Request_id is sent like address data (message parts that can be read using XREP sockets only, and finished by empty message). After address data configured parts of the request are sent one by one as multipart message. E.g. if you have following in configuration:

```
zmq-forward:
  contents:
    - !Method
    - !Uri
    - !Header Cookie
    - !Body
```

And you've got request:

```
POST /hello HTTP/1.1
Host: example.com
Cookie: example=cookie_value
Content-Length: 8

PostBody
```

You will receive following message parts (one line per message part):

```
POST
/hello
example=cookie_value
PostBody
```

It's up to the application for how to act upon it. Note if you set `retry` to something you can get same request twice. And if `retry` is set to `!RetryFirst <N>` request id will be same for every attempt, if you've set `retry` to `!RetryLast <N>` request id will change on each attempt. But usually request id is opaque for user in zeromq.

2.1.2 Response

Response can contain one, two or three parts for convenience.

In the simple case you just send message body, as a single part message. Zerogw will respond with `200 OK` and that message body.

If you respond with two messages first one will be status line, so yo can respond with 404 page like the following:

```
404 Not Found
<h1> Page Not Found</h1>
```

Note: These ways are quite unuseful in real situations. `Content-Length` header will be added automatically, but you should configure specific `Content-Type` header in a config to be sure that browser will render page correctly when using this method

---

If you need to supply headers you send 3 message parts. Second one is constructed from nul-terminated name/value header pairs:

```
200 OK
Content-Type\0text/html\0E-tag\0immortal\0
<b>Lorem ipsum dolor sit amet</b>
```

Note: Last header value must be nul-terminated. You must not add `Content-Length` header as it will be generated automatically. Currently headers sent from backend will be appended to headers specified in config without overwriting, it can lead to unexpected behavior on some proxies or browsers so you should use use one or another way for each header type throught the whole application.

2.2 WebSockets Backend Protocol

Zerogw implements unified interface for application writers for both long polling and websockets. Both are used for bidirectional message channels from client to server.

Note: There is no overhead of using long polling with normal http backend in zerogw if that suits your application. This interface is provided to make using either websockets or long polling transparent for both frontend and backend developer and provides reliable message stream.

2.2.1 Zerogw to Backend Messages

Most messages from server to client consists of client id (long binary string of nonsense) and ascii command name, following more message parts which we will call arguments in the text below.

Connection Messages

`connect` is sent when new connection established, no arguments

`disconnect` is sent when connection disconnected. All subscriptions (see below) are already cancelled so you don't need to remove them, but you can cleanup some application-specific data. No arguments

Starting with v0.5.10: disconnect appends an cookie (see below) as an argument, if cookie is set (it breaks compatibility somewhat with versions starting with v0.5.8, which did not return cookie on the disconnect)

Messages

`message` message sent from frontend to websocket, has single argument - message text. Can be binary if the browser (or malicious client) sent binary data

`msgfrom` message sent from frontend to websocket, has two arguments *cookie* and *message text*, latter is same as in `message` and former is an opaque string set by `set_cookie` (see below)

Heartbeats

There are two kinds of heartbeat messages:

- plain heartbeat, activated by `heartbeat-interval` setting
- synchronisation message containing connection ids, activated by `sync-interval` setting

Both start with `server id` message. For the former server id is followed by literal ascii `heartbeat`. Latter consists of literal ascii `sync` followed by pairs connection_id, cookie (latter is empty if cookie is not sent, but is always sent).

Sync messages are only sent to named outputs (see below), and can be used to synchronize user list with backend in case of network failures (`connect` or `disconnect` message lost), backend failures (could not process `connect` or `disconnect` message, because backend crashed when processing message) or zerogw crashes (zerogw can't send `disconnect` messages after restart).

2.2.2 Backend to Zerogw Messages

Usually messages sent from backend are published using pubsub to several zerogw. This allows not to track where user currently is and also allows to publish messages to several users without doing that on backend.

Direct Messages

`send, conn_id, message` sends message directly to the user. You can send binary message, but most browsers can read only text data, so use utf-8

`sendall, message` sends message to all connections. Of course addressees are limited to a single route, not to the whole zerogw. Note that message is also sent to unauthenticated connections. You need to subscribe all users to some topic and use `publish` if you want to send to authenticated users only.

Topic Subscription

Topics is a mechanism in zerogw which allows you to send message to several users effeciently. You first subscribe users to a topic, send publish a message to a topic, and all users get this message. Topic is an opaque binary string. Topics are created and removed on demand and are quite fast to use them for a lot of things.

`subscribe, conn_id, topic` subscribes connection

`unsubscribe, conn_id, topic` unsubscribes connection

`publish, topic, message` publish message to a topic, message will be delivered to all connections subscribed to the topic

`clone, source_topic, target_topic` clones subscriptions of `source_topic` such as all its connections are now subscribed to both topics, connections that where subscribed to `target_topic` are left intact (so it can be thought as merge operation)

drop, *topic* delete topic, unsubscribing all the users (can be combined with `clone` to achieve "rename" or "join" effect)

Outputs

In addition to subscription clients on topics you can subscribe subset of client messages to a specific named backend (`named-outputs` in config)

add_output, *conn_id*, *msg_prefix*, *name* map prefix to specific output

del_output, *conn_id*, *msg_prefix* unmap prefix

As with subscriptions don't need to unmap anything from disconnected user.

Note: it's your responsibility to clean user state from the backend. `disconnect` messages are sent to main backend only

---

Cookie

Cookie is a experimental feature of zerogw v0.5.8, which allows to prepend opaque data to all messages sent from a client. This is usually used to authorize connection without need to access authorization database on each user's message. Only one cookie can be attached at a time, but you can change the cookie at any time. Once set, you can't discard cookie. Once cookie attached all messages will be forwarded using `msgfrom` message type with cookie and data.

set_cookie, *conn_id*, *cookie* set cookie for the connection, cookie is an opaque string

---

Note: starting with v0.5.10 cookie set with `set_cookie` are sent in `disconnect` messages. Since disconnect can occur before you were able to set cookie you must tolerate different number of arguments in `disconnect` messsages.

CHAPTER 3

Control Socket Protocol

CHAPTER 4

HTTP Tutorial

4.1 Disclaimer

Unlike most HTTP servers out there zerogw doesn't try to support old CGI convention of passing environment to your web application. For each forwarded requests zerogw forwards only a few fields, such as url and method (configurable) to a backend. This way we produce less internal traffic and less bloated applications. This also means thay you can't use most of bloated web frameworks out there. Zerogw is suitable for fast web applications (and a parts of thereof) which need most out of machine performance (e.g. you can use it for autocompletion or some other AJAX). If you want to use big fat web framework there are plenty of other solutions. If thats ok for you, read on!

4.2 Hello World

Let's start with simple hello world application written in python. The first thing to know is how to configure zerogw. We will start with simplest possible configuration and will improve it later.

All configuration settings should be written into a separate file with the YAML convention. Here we call it zerowg.yaml.

Minimal configuration:

```
Server:
  listen:
  - host: 0.0.0.0
    port: 8080

Routing:
  zmq-forward:
    enabled: yes
    socket:
    - !zmq.Bind "tcp://127.0.0.1:5000"
    contents:
    - !Uri
```

All above means that zerogw will listen for connections on port 8080 on all interfaces (0.0.0.0). Then it will forward requests to local host with port 5000. This port also listens for connections, so you can start several backend processes (and even several boxes, if you'll change 127.0.0.1 to your local network ip address) to process requests. Forwarded request will contain just URI part of the original request.

Then we will write the simple python script which would make this work:

```
import zmq

ctx = zmq.Context()
sock = ctx.socket(zmq.REP)
sock.connect('tcp://127.0.0.1:5000')
while True:
    uri, = sock.recv_multipart()
    sock.send_multipart([b'Hello from '+uri])
```

Next start the zerowg server and use -c to tell zerogw the configuration file:

```
zerogw -c ./zerogw.yaml
```

Open a new terminal and start your python script:

```
python ./ourserver.py
```

This is everything which is needed to serve requests. Note we are connecting to the address you specified to bind to in zerogw.yaml.

Now you can go to the browser at http://localhost:8080/ and you should see `Hello from /`.

We use `recv_multipart` and `send_multipart` to simplify working with sockets. If they are not provided in your language bindings you will probably need to use `recv` and `send` while reading `RCVMORE` and setting `SNDMORE` flags. Refer to zeromq and you language bindings for more information.

Note: This code works perfectly for example, but in reality it can except suring recv or send calls. So in production application you should use more complicated loop. See ioloop in pyzmq bindings or appropriate functionality in your language bindings

---

Was that hard? I guess no really.

CHAPTER 5

WebSocket Tutorial

5.1 Overview

Zerogw supports unified interface for both websockets and websocket emulation for long polling. This tutorial will guide you throught the process of creating simple near real-time web chat with zerogw and few dozens lines of backend code.

CHAPTER 6

Configuration Guide

CHAPTER 7

Indices and tables

- *genindex*
- *search*