Boreas Documentation

Release 1.0.0

Karol Majta

Sep 27, 2017

Contents

Contents:

CHAPTER 1

Quickstart

What is Boreas

Boreas is a websocket server implementing a pub/sub functionality. It works out of the box.

It supports:

- backend agnostic, token based authentication of subscribers
- channels accessible for subscribers via. WebSockets
- publishing via HTTP api
- server state polling via HTTP api

If you're looking for a lightweight solution for typical pub/sub pattern implementations (chatrooms, live feeds etc.) it may suit your needs just right.

Installing

Boreas is available from Cheeseshop:

```
$ pip install boreas
```

You can always get the latest version from GitHub:

```
$ git clone https://github.com/lolek09/boreas.git
$ cd boreas
$ pip install .
```

Running

Running boreas with default settings is simple:

```
$ boreas
Using `boreas.utils.tokens:no_tokens` to get authentication tokens...
Done. Loaded 0 tokens.
Configuring servers...
Running in debug mode. Available debug urls are:
        - /debug/token-dump
        - /debug/recipient-dump
API endpoint configured to listen on 127.0.0.1:8001
WebSocket endpoint configured to listen on 127.0.0.1:8002
Running...
```

Default configuration is reasonable for development setup.

Baby steps

To check out what is possible with Boreas we will use two tools - **curl** and **wsdump.py**. The first one is probably available on your system, the second one comes with *websocket-client* package. You can easily get it:

```
$ pip install websocket-client
```

Token creation

By default Boreas starts with no authentication tokens. It is possible to provide them at startup or add them at runtime using the HTTP api. Let's use the second method, to create two tokens for further use:

```
$ curl -X POST http://localhost:8001/token/
{"token":
↪"I4myiMMyKWLBKaLiuKKHQTk1mKfkZrmBEGlMphbVPSucBTMCTbZaz1wlyVm0l6yPUnKK10B1ZFWQvUGpqwqx2Mx9ho0At42kpW
↪1361390859"}
curl -X POST http://localhost:8001/token/
{"token":
↪"WJIL3PnxrEZu2F8WGJ3T2bevcGSF6jrmCQZlRUhv44sFvM4qe5iYcHO69bWdKiUYZWhril002HkIa6W0NTJqPFOkzeZ7XM1aXm
↪1361390902"}
```

You will be able to use these two tokens to authenticate two users. In debug mode it's actually possible to verify that these tokens were created:

```
$ curl  http://localhost:8001/debug/token-dump
I4myiMMyKWLBKaLiuKKHQTk1mKfkZrmBEGlMphbVPSucBTMCTbZaz1wlyVm0l6yPUnKK10B1ZFWQvUGpqwqx2Mx9ho0At42kpWmxs
↪1361390859
WJIL3PnxrEZu2F8WGJ3T2bevcGSF6jrmCQZlRUhv44sFvM4qe5iYcHO69bWdKiUYZWhril002HkIa6W0NTJqPFOkzeZ7XM1aXm2Fq
↪1361390902
```

Subscriber authentication

Ok, now it's time for some subscriber action. In another terminal window run:

```
$ wsdump.py ws://localhost:8002
```

You're now the websocket client, and your status is *anonymous*. You can check it using debug url:

```
$ curl http://localhost:8001/debug/recipient-dump
RECEIVER POOL REPORT
anonymous: 0
authenticated:
        ---
channels:
        ---
```

The only valid message you can send right now is the authentication message. Until you authenticate, you won't be receiving any messages. So, Just use one of your tokens and authenticate:

```
> {"access_token": "I4myiMMyKWLBKaLiuKKHQT..."}
```

You won't get any response but you can check debug url again to see if you were authenticated.

Subscribing and unsubscribing channels

After you authenticated you are able to subscribe and unsubscribe channels. You can subscribe virtually any channel name, however it is not guaranteed that anything will ever get published on it. It is your job, to keep your subscribers informed on which channels are available.

Let's make use of the fact that we can subscribe any channel, and make their names easy to remember:

```
> {"channels": {"join": ["ch1", "ch2"], "leave": []}}
```

In channels property you specify list of channel names you want to join, and ones you want to leave. Note that both fields are required. If you don't want to join or leave any channels just supply an empty list.

Use debug urls to verify if everything went fine.

Publishing to channels

After you have a subscriber, you can use the HTTP api to send broadcasts:

```
> curl -X POST -H "Content-Type: application/json" http://localhost:8001/broadcast/
↪ch1/ -d '{"dryrun": false, "payload": {}}'
```

Boom! Check out the Websocket terminal for epic results. You can fiddle some more with the server trying to send broadcasts and subsciribe/unsubscribe channels. Then feel free to check out rest of the documentation.

MORE DOCS ARE COMING

CHAPTER 2

Indices and tables

- genindex
- modindex
- search