
Django-tornado-websockets Documentation

Release 0.2.0

Hugo ALLIAUME

November 01, 2016

1	Architecture	3
2	Documentation	5
2.1	Installation	5
2.1.1	Automatic installation	5
2.1.2	Manual installation	5
2.2	Django integration and configuration	5
2.2.1	Integration	5
2.2.2	Configuration	6
2.3	Usage	7
2.3.1	Run Tornado server	7
2.3.2	Using WebSockets (server side)	8
2.3.3	Using WebSockets (client side)	9
2.4	Modules	11
2.4.1	Module	11
2.4.2	Progress bar	11
2.5	API	13
2.5.1	WebSocket	13
2.5.2	WebSocketHandler	14
2.5.3	TornadoWrapper	14
	Python Module Index	17

Django-tornado-websockets is a useful solution to provide an easy way to use [Tornado WebSockets](#) with a [Django](#) application.

Important: Django-tornado-websockets is actually in **alpha version**!

Architecture

Fig. 1.1: Example of an architecture using Tornado as WSGI server, Django and django-tornado-websockets

To use django-tornado-websockets's WebSockets, you should use **Tornado as a WSGI server** where you will define handlers to *handle* an incoming request. Since we already have a WSGI server, it's probably useless to try running **Gunicorn** or **uWSGI** as WSGI server. You can try to wrap Tornado **WSGI server** into Gunicorn/uWSGI **WSGI server** but... It's a bit retarded I think (___

Let's explain this diagram:

1. The client make a request to our web server with his web browser,
2. Our web server (**nginx**, **Apache**, ...) pass this WSGI or WebSocket request to Tornado ¹,
3. If it is a *WebSocket request*, we pass it to `tornado.websocket`, otherwise it's Django that will handle this request,
4. We wait for a *WebSocket response* or a Django response,
5. and 6. Then we return this response to the client.

¹ I forgot it on the diagram, but nginx or Apache has the job to deliver static files, it's not Tornado's work

2.1 Installation

2.1.1 Automatic installation

```
$ pip install django-tornado-websockets

# In your Django project
$ python manage.py collectstatic
```

2.1.2 Manual installation

```
$ git clone --recursive https://github.com/Kocal/django-tornado-websockets.git
$ cd django-tornado-websockets
$ python setup.py install

# In your Django project
$ python manage.py collectstatic
```

2.2 Django integration and configuration

- *Integration*
- *Configuration*
 - *Basic configuration*
 - *Django support*
 - *Static files support*
 - *Additional settings*

2.2.1 Integration

In your `settings.py` file, you need to add `tornado_websockets` to your Django `INSTALLED_APPS` :

```
INSTALLED_APPS = [
    # ...
    'tornado_websockets',
]
```

2.2.2 Configuration

Since we use Tornado as a replacement of a WSGI server (Gunicorn, uWSGI, ...), you need to configure it a bit before using `django-tornado-websockets`.

Basic configuration

You can provide a Tornado configuration in your `settings.py` file like this:

```
# At the end of settings.py file

TORNADO = {
    'port': 1337,      # 8000 by default
    'handlers': [],   # [] by default
    'settings': {},   # {} by default
}
```

1. `port` is the port which Tornado main loop will listen for its `HTTPServer`,
2. `handlers` is a list of tuples where you can make a link between a route and an handler,
3. `settings` is a dictionary used to customize various aspects of Tornado (autoreload, debug, ...).

Read more about Tornado handlers and settings in the Tornado documentation: [Application configuration](#)

Django support

To makes Django work with Tornado, you need to add a new handler to Tornado configuration. Tornado can [runs WSGI apps](#) (like Django) by using `tornado.wsgi.WSGIContainer`, and we provide an already defined Django WSGI app that you can easily use.

You can also make your own Django WSGI app using the `tornado_websockets/__init__.py` file.

```
import tornado_websockets

# ...

TORNADO = {
    # ...
    'handlers': [
        # ...
        tornado_websockets.django_app(),  # django_app is using a "wildcard" route, so it should be
    ],
}
```

Static files support

If you need static files support during your development (so you are not running a configured nginx/Apache for static files), you can add another handler to your configuration:

```
import tornado.web

# ...

# Django specific configuration about static files
STATIC_URL = '/static/'
STATIC_ROOT = os.path.join(BASE_DIR, 'static')

TORNADO = {
    # ...
    'handlers': [
        (r'%s(.*)' % STATIC_URL, tornado.web.StaticFileHandler, {'path': STATIC_ROOT}),
        # ...
    ]
}
```

Additional settings

You can pass additional settings to Tornado with `TORNADO['settings']` dictionary. For example, it can be useful to set `'debug': True` if you are still in a development phase:

```
TORNADO = {
    # ...
    'settings': {
        'debug': True,
    }
}
```

2.3 Usage

- *Run Tornado server*
- *Using WebSockets (server side)*
 - *Create a WebSocket application*
 - *Receive an event from a client*
 - *Send an event to a client*
- *Using WebSockets (client side)*
 - *Linking JS wrapper into your Django template*
 - *Create a WebSocket connection*
 - *Receive an event from the server*
 - *Send an event to the server*

2.3.1 Run Tornado server

Django-tornado-websockets provides an easy solution to run your Tornado server. When you add `tornado_websockets` to your `INSTALLED_APPS`, you obtain a new management command called `runornado`:

```
$ python manage.py runornado
```

2.3.2 Using WebSockets (server side)

It's preferable to write your WebSocket applications in your `views.py` file, or import these in `views.py`.

Create a WebSocket application

You should use the `WebSocket` class to use... WebSockets . It takes only one parameter and it's the path. This path should be unique because it's automatically adding a new handler to Tornado handlers (`your_path <=> your_websocket`):

```
from tornado_websockets.websocket import WebSocket

# Make a new instance of WebSocket and automatically add handler '/ws/my_ws' to Tornado handlers
my_ws = WebSocket('/my_ws')
```

Note: If you are using this decorator on a class method (a wild `self` parameter appears!), you need to define a context for the `WebSocket` instance because `@my_ws.on` decorator can not know by itself what value `self` should take (or maybe I am doing it wrong?):

```
class MyClass(object):

    def __init__(self):
        my_ws.context = self
```

Receive an event from a client

To listen an incoming event, you should use the `@my_ws.on` decorator (where `my_ws` is an instance of `WebSocket`) on a function or a class method.

Functions and class methods **should take two named parameters**:

- `socket`: client who sent the event (instance of `WebSocketHandler`),
- `data`: data sent by the client (dictionary).

Usage example:

```
# On a function
@my_ws.on
def my_event(socket, data):
    print('Catch "my_event" event from a client')
    print('But I know this client, it is the one using this websocket connection: %s' % socket)

# On a class method
class MyClass(object):

    def __init__(self):
        # Do not forget the context, otherwise the `self` value for all class methods decorated by `
        # decorator will be `None`
        my_ws.context = self

    @my_ws.on
    def my_other_event(self, socket, data):
        # `self` value is a MyClass instance due to `my_ws.context = self` in `__init__()` method
```

```
print('Catch "my_other_event" from a client')
print('And same as before, I know that this client is using this websocket connection: %s' %
```

Send an event to a client

Warning: You can only emit an event in a function or method decorated by `@my_ws.on` decorator.

There is three ways to emit an event:

1. For **all clients connected to your WebSocket application**, you should use `my_ws.emit` method,
2. For **the client who just sent an event**, you should use `socket.emit` method,
3. For **a specific client**, it's not officially implemented but you can take a look at `my_ws.handlers`. It's a `WebSocketHandler` list and represents all clients connected to your application, so you can use `my_ws.handlers[0].emit` method.

Usage example (echo server):

```
from tornado_websockets.websocket import WebSocket

ws_echo = WebSocket('/echo')

@ws_echo.on
def open(socket):
    # Notify all clients about a new connection
    ws_echo.emit('new_connection')

@ws_echo.on
def message(socket, data):
    # Reply to the client
    socket.emit('message', data)

    # Wow we got a spammer, let's inform the first client :^)
    if 'spam' in data.message:
        # wow
        ws_echo[0].emit('got_spam', {
            'message': data.get('message'),
            'socket': socket
        })
```

For more examples, you can read `testapp/views.py` file.

2.3.3 Using WebSockets (client side)

Django-tornado-websockets uses its own wrapper for using JavaScript WebSocket in client-side: `django-tornado-websockets-client`. By using this wrapper, you will be able to write:

```
var ws = new TornadoWebSocket(...);

ws.on('open', () => {
    ws.emit('my_event', { foo: 'bar' });
});

// instead of
var ws = new WebSocket(...);
```

```
ws.onopen = () => {
    ws.send({ event: 'my_event', data: { foo: 'bar' } });
};
```

But you can simply ignore this wrapper and use `raw WebSocket` if you want. Just remember that data passed by Django-tornado-websockets are in JSON: `{event: 'evt', data: {}}`.

Linking JS wrapper into your Django template

Link `django-tornado-websockets-client.js` (symbolic link to `main.min.js`) file in your Django template:

```
{% load static %}
<script src="{% static 'tornado_websockets/client.js' %}"></script>
```

Create a WebSocket connection

There is three ways to create a WebSocket connection:

```
var ws = new TornadoWebSocket(path, options);
var ws = TornadoWebSocket(path, options); // shortcut to new TornadoWebSocket(path, options)
var ws = tws(path, options); // shortcut to new TornadoWebSocket(path, options)
```

class TornadoWebSocket (*String path, Object options*)

Initialize a new WebSocket object with given options.

Parameters:

- `path`: same value than `path` parameter from *WebSocket*, see [create websocket application](#),
- `options.host`: host used for connection (default: 'localhost', but soon `window.location`)
- `options.port`: port used for connection (default: 8000)
- `options.secure`: true for using a secure connection (default: false)

Receive an event from the server

You can listen to WebSocket's events `onopen`, `onclose` and `onerror` (`onmessage` too but you will rewrite a core part). You can also listen to your own events like `my_event`, `user_joined`, etc...

TornadoWebSocket.on (*String event, Function callback*)

Bind a function to an event.

Parameters:

- `event`: Event name
- `callback`: Function to execute when event event is received

```
// Bind to WebSocket.onopen
ws.on('open', event => {
    console.log('Connection: OPEN', event);

    // Add an event/callback combination into TornadoWebSocket.events private object.
    // Will be called when we receive a JSON like that: {event: 'my_event', data: {...}}
    ws.on('my_event', data => {
```

```

        console.log('Got data from « my_event »', data);
    });
});

// Bind to WebSocket.onclose
ws.on('close', event => {
    console.log('Connection: ERROR', event);
});

// Bind to WebSocket.onerror
ws.on('error', event => {
    console.log('Connection: CLOSED', event);
});

```

Send an event to the server

TornadoWebSocket.**emit** (*String event, Object|* data*)

Send a pair event/data to the server.

Parameters:

- **event**: Event name
- **data**: Data to send, can be an Object, not an Object (will be replaced by { data: { message: data } }, or undefined (will be replaced by {}))

```

ws.on('open', event => {
    ws.emit('my_event'); // will send {}

    ws.emit('my_event', 'My message'); // will send {message: 'My message'}

    ws.emit('my_event', {my: 'data'}); // will send {my: 'data'}
});

```

2.4 Modules

2.4.1 Module

class tornado_websockets.modules.**Module** (*name=''*)

2.4.2 Progress bar

The module « ProgressBar » facilitate the communication between the server-side and client-side of a progression bar.

Server-side:

- An easier communication with client-side ProgressBar module
- Handle *init*, *update* and *done* events,
- Update current progression value with *tick()* or *reset()*

Client-side:

- An easier communication with server-side ProgressBar module,

- Handle *init*, *update* and *done* events,
- Rendering a progression bar by using *HTML5* or *Bootstrap* rendering.

Server-side

Construction

class `tornado_websockets.modules.ProgressBar` (*name*='', *min*=0, *max*=100, *indeterminate*=False)

Initialize a new `ProgressBar` module instance.

If *min* and *max* values are equal, this progress bar has its indeterminate state set to `True`.

Parameters

- **min** (*int*) – Minimum value
- **max** (*int*) – Maximum value

Methods

`ProgressBar.reset()`

Reset progress bar's progression to its minimum value.

`ProgressBar.tick(label=None)`

Increments progress bar's `_current` by 1 and emit `update` event. Can also emit `done` event if progression is done.

Call `emit_update()` method each time this method is called. Call `emit_done()` method if progression is done.

Parameters **label** (*str*) – A label which can be displayed on the client screen

`ProgressBar.is_done()`

Return `True` if progress bar's progression is done, otherwise `False`.

Returns `False` if progress bar is indeterminate, returns `True` if progress bar is determinate and current value is equals to *max* value. Returns `False` by default.

Return type `bool`

Events

`ProgressBar.on(callback)`

Shortcut for `tornado_websockets.websocket.WebSocket.on()` decorator, but with a specific prefix for each module.

Parameters **callback** (*Callable*) – function or a class method.

Returns `callback` parameter.

`ProgressBar.emit_init()`

Emit `before_init`, `init` and `after_init` events to initialize a client-side progress bar.

If progress bar is not indeterminate, *min*, *max* and *value* values are sent with `init` event.

`ProgressBar.emit_update(label=None)`

Emit `before_update`, `update` and `after_update` events to update a client-side progress bar.

Parameters `label (str)` – A label which can be displayed on the client screen

`ProgressBar.emit_done()`

Emit done event when progress bar's progression is done().

Example

```
from tornado import gen

from tornado_websockets.modules import ProgressBar
from tornado_websockets.websocket import WebSocket

ws = WebSocket('module_progressbar')
progressbar = ProgressBar('foo', min=0, max=100)

ws.bind(progressbar)

@progressbar.on
def reset():
    progressbar.reset()

@progressbar.on
@gen.engine # Make this function asynchronous for Tornado's IOLoop
def start():
    for value in range(0, progressbar.max):
        yield gen.sleep(.1) # like time.sleep(), but asynchronous with @gen.engine
        progressbar.tick(label="[%d/%d] Tâche %d terminée" % (progressbar.current + 1, progressbar.m
```

Client-side

Read documentation about `ProgressBar` client-side module [here](#).

2.5 API

2.5.1 WebSocket

class `tornado_websockets.websocket.WebSocket (path)`

Class that you should to make `WebSocket` applications .

`WebSocket.on (callback)`

Should be used as a decorator.

It will execute the decorated function when `WebSocketHandler` will receive an event where its name correspond to the function one (by using `__name__` magic attribute).

Parameters `callback (callable)` – Function to decorate.

Raises `tornado_websockets.exceptions.NotCallableError` –

Example

```
>>> ws = WebSocket('/example')
>>> @ws.on
... def hello(socket, data):
...     print('Received event « hello » from a client.')
```

`WebSocket.emit(event, data=None)`

Send an event/data dictionary to all clients connected to your `WebSocket` instance. To see all ways to emit an event, please read « [Send an event to a client](#) » section.

Parameters

- **event** (*str*) – event name
- **data** (*dict or str*) – a dictionary or a string which will be converted to `{'message': data}`

Raise `EmitHandlerError` if not used inside `@WebSocket.on()` decorator.

Raise `tornado.websocket.WebSocketClosedError` if connection is closed.

Warning: `WebSocket.emit()` method should be used inside a function or a class method decorated by `@WebSocket.on()` decorator, otherwise it will raise a `EmitHandlerError` exception.

2.5.2 WebSocketHandler

`class tornado_websockets.websockethandler.WebSocketHandler(application, request, **kwargs)`

Represents a `WebSocket` connection, wrapper of `tornado.websocket.WebSocketHandler` class.

This class should not be instantiated directly; use the `WebSocket` class instead.

`WebSocketHandler.initialize(websocket)`

Called when class initialization, makes a link between a `WebSocket` instance and this object.

Parameters `websocket` (`WebSocket`) – instance of `WebSocket`.

`WebSocketHandler.on_message(message)`

Handle incoming messages on the `WebSocket`.

Parameters `message` (*str*) – JSON string

`WebSocketHandler.on_close()`

Called when the `WebSocket` is closed, delete the link between this object and its `WebSocket`.

`WebSocketHandler.emit(event, data)`

Sends a given event/data combinaison to the client of this `WebSocket`.

Wrapper for `tornado.websocket.WebSocketHandler.write_message` method.

Parameters

- **event** (*str*) – event name to emit
- **data** (*dict*) – associated data

2.5.3 TornadoWrapper

`class tornado_websockets.tornadowrapper.TornadoWrapper`

Wrapper for Tornado application and server handling.

It let you access to Tornado app, handlers and settings everywhere in your code (it's really useful when you run `run tornado` management command and WebSockets management).

classmethod `TornadoWrapper.add_handler(handler)`

Add an handler to Tornado app if it's defined, otherwise it add this handler to the `TornadoWrapper.tornado_handlers` list.

Parameters `handler` (*list/tuple*) – An handler which conforms

classmethod `TornadoWrapper.start_app(handlers=None, settings=None)`

Initialize the Tornado web application with given handlers and settings.

Parameters

- **handlers** (*list*) – Handlers (routes) for Tornado
- **settings** (*dict*) – Settings for Tornado

Returns `None`

classmethod `TornadoWrapper.loop()`

Run Tornado main loop and display configuration about Tornado handlers and settings.

Returns `None`

classmethod `TornadoWrapper.listen(tornado_port)`

Start the Tornado HTTP server on given port.

Parameters `tornado_port` (*int*) – Port to listen

Returns `None`

Todo

Add support for HTTPS server.

t

`tornado_websockets.modules`, [11](#)
`tornado_websockets.tornadowrapper`, [14](#)
`tornado_websockets.websocket`, [13](#)
`tornado_websockets.websockethandler`, [14](#)

A

`add_handler()` (`tornado_websockets.tornadowrapper.TornadoWrapper` class method), 15

E

`emit()` (`tornado_websockets.websocket.WebSocket` method), 14

`emit()` (`tornado_websockets.websockethandler.WebSocketHandler` method), 14

`emit_done()` (`tornado_websockets.modules.ProgressBar` method), 13

`emit_init()` (`tornado_websockets.modules.ProgressBar` method), 12

`emit_update()` (`tornado_websockets.modules.ProgressBar` method), 12

I

`initialize()` (`tornado_websockets.websockethandler.WebSocketHandler` method), 14

`is_done()` (`tornado_websockets.modules.ProgressBar` method), 12

L

`listen()` (`tornado_websockets.tornadowrapper.TornadoWrapper` class method), 15

`loop()` (`tornado_websockets.tornadowrapper.TornadoWrapper` class method), 15

M

`Module` (class in `tornado_websockets.modules`), 11

O

`on()` (`tornado_websockets.modules.ProgressBar` method), 12

`on()` (`tornado_websockets.websocket.WebSocket` method), 13

`on_close()` (`tornado_websockets.websockethandler.WebSocketHandler` method), 14

`on_message()` (`tornado_websockets.websockethandler.WebSocketHandler` method), 14

P

`ProgressBar` (class in `tornado_websockets.modules`), 12

R

`reset()` (`tornado_websockets.modules.ProgressBar` method), 12

S

`start_app()` (`tornado_websockets.tornadowrapper.TornadoWrapper` class method), 15

T

`tick()` (`tornado_websockets.modules.ProgressBar` method), 12

`tornado_websockets.modules` (module), 11

`tornado_websockets.tornadowrapper` (module), 14

`tornado_websockets.websocket` (module), 13

`tornado_websockets.websockethandler` (module), 14

`TornadoWebSocket` (class), 10

`TornadoWebSocket.emit()` (`TornadoWebSocket` method), 11

`TornadoWebSocket.on()` (`TornadoWebSocket` method), 10

`TornadoWrapper` (class in `tornado_websockets.tornadowrapper`), 14

W

`WebSocket` (class in `tornado_websockets.websocket`), 13

`WebSocketHandler` (class in `tornado_websockets.websockethandler`), 14