



# Getting Started

## What is Socket.IO?

Socket.IO is a transport protocol that enables real-time bidirectional event-based communication between clients (typically, though not always, web browsers) and a server. The official implementations of the client and server components are written in JavaScript. This package provides Python implementations of both, each with standard and `asyncio` variants.

## Version compatibility

The Socket.IO protocol has been through a number of revisions, and some of these introduced backward incompatible changes, which means that the client and the server must use compatible versions for everything to work.

If you are using the Python client and server, the easiest way to ensure compatibility is to use the same version of this package for the client and the server. If you are using this package with a different client or server, then you must ensure the versions are compatible.

The version compatibility chart below maps versions of this package to versions of the JavaScript reference implementation and the versions of the Socket.IO and Engine.IO protocols.

JavaScript Socket.IO version	Socket.IO pro- tocol revision	Engine.IO pro- tocol revision	python-sock- etio version	python-en- gineio version
0.9.x	1, 2	1, 2	Not supported	Not supported
1.x and 2.x	3, 4	3	4.x	3.x
3.x and 4.x	5	4	5.x	4.x

## Client Examples

The example that follows shows a simple Python client:

```
import socketio

sio = socketio.Client()

@sio.event
def connect():
    print('connection established')

@sio.event
def my_message(data):
    print('message received with ', data)
    sio.emit('my response', {'response': 'my response'})

@sio.event
def disconnect():
    print('disconnected from server')

sio.connect('http://localhost:5000')
sio.wait()
```

Below is a similar client, coded for `asyncio` (Python 3.5+ only):

```
import asyncio
import socketio
```



```

sio = socketio.AsyncClient()

@sio.event
async def connect():
    print('connection established')

@sio.event
async def my_message(data):
    print('message received with ', data)
    await sio.emit('my response', {'response': 'my response'})

@sio.event
async def disconnect():
    print('disconnected from server')

async def main():
    await sio.connect('http://localhost:5000')
    await sio.wait()

if __name__ == '__main__':
    asyncio.run(main())

```

## Client Features

- Can connect to other Socket.IO servers that are compatible with the JavaScript Socket.IO reference server.
- Compatible with Python 3.8+.
- Two versions of the client, one for standard Python and another for asyncio.
- Uses an event-based architecture implemented with decorators that hides the details of the protocol.
- Implements HTTP long-polling and WebSocket transports.
- Automatically reconnects to the server if the connection is dropped.

## Server Examples

The following application is a basic server example that uses the Eventlet asynchronous server:

```

import eventlet
import socketio

sio = socketio.Server()
app = socketio.WSGIApp(sio, static_files={
    '/': {'content_type': 'text/html', 'filename': 'index.html'}
})

@sio.event
def connect(sid, environ):
    print('connect ', sid)

@sio.event
def my_message(sid, data):
    print('message ', data)

@sio.event
def disconnect(sid):
    print('disconnect ', sid)

if __name__ == '__main__':
    eventlet.wsgi.server(eventlet.listen(('', 5000)), app)

```

Below is a similar application, coded for `asyncio` (Python 3.5+ only) and the Uvicorn web server:

```

from aiohttp import web
import socketio

```



```

sio = socketio.AsyncServer()
app = web.Application()
sio.attach(app)

async def index(request):
    """Serve the client-side application."""
    with open('index.html') as f:
        return web.Response(text=f.read(), content_type='text/html')

@sio.event
def connect(sid, environ):
    print("connect ", sid)

@sio.event
async def chat_message(sid, data):
    print("message ", data)

@sio.event
def disconnect(sid):
    print('disconnect ', sid)

app.router.add_static('/static', 'static')
app.router.add_get('/', index)

if __name__ == '__main__':
    web.run_app(app)

```

## Server Features

- Can connect to servers running other Socket.IO clients that are compatible with the JavaScript reference client.
- Compatible with Python 3.8+.
- Two versions of the server, one for standard Python and another for asyncio.
- Supports large number of clients even on modest hardware due to being asynchronous.
- Can be hosted on any [WSGI](#) or [ASGI](#) web server including [Gunicorn](#), [Uvicorn](#), [eventlet](#) and [gevent](#).
- Can be integrated with WSGI applications written in frameworks such as Flask, Django, etc.
- Can be integrated with [aiohttp](#), [FastAPI](#), [sanic](#) and [tornado](#) [asyncio](#) applications.
- Broadcasting of messages to all connected clients, or to subsets of them assigned to “rooms”.
- Optional support for multiple servers, connected through a messaging queue such as Redis or RabbitMQ.
- Send messages to clients from external processes, such as Celery workers or auxiliary scripts.
- Event-based architecture implemented with decorators that hides the details of the protocol.
- Support for HTTP long-polling and WebSocket transports.
- Support for XHR2 and XHR browsers.
- Support for text and binary messages.
- Support for gzip and deflate HTTP compression.
- Configurable CORS responses, to avoid cross-origin problems with browsers.

**Monetize your audience:** Fund an OSS project or website with EthicalAds, a [privacy-first ad network](#)

Ads by EthicalAds

