**Websockets in Python**

[[Gaurav Kumar](https://gaurav-adarshi.medium.com/?source=post_page-----e8f845d52640--------------------------------)](https://gaurav-adarshi.medium.com/?source=post_page-----e8f845d52640--------------------------------)

[[Stackademic](https://blog.stackademic.com/?source=post_page-----e8f845d52640--------------------------------)](https://blog.stackademic.com/?source=post_page-----e8f845d52640--------------------------------)

[Gaurav Kumar](https://gaurav-adarshi.medium.com/?source=post_page-----e8f845d52640--------------------------------)

·

[Follow](https://medium.com/m/signin?actionUrl=https%3A%2F%2Fmedium.com%2F_%2Fsubscribe%2Fuser%2Ff319c9e06fc1&operation=register&redirect=https%3A%2F%2Fblog.stackademic.com%2Fwebsockets-in-python-e8f845d52640&user=Gaurav+Kumar&userId=f319c9e06fc1&source=post_page-f319c9e06fc1----e8f845d52640---------------------post_header-----------)

Published in

[Stackademic](https://blog.stackademic.com/?source=post_page-----e8f845d52640--------------------------------)

·

4 min read

·

Jan 8

147

**What is WebSockets?**

WebSockets provide a bidirectional communication channel over a single, long-lived connection between clients and servers. Unlike HTTP, which is stateless and requires a new connection for each request-response cycle, WebSockets allow continuous, real-time communication between a client and a server. In Python, the websockets library offers a simple and efficient way to work with WebSockets.

**How WebSockets Work?**

**Here are five main points on how WebSockets work:**

**Connection Establishment:**

* WebSockets begin with a handshake process between the client and server, initiated through an HTTP request. This handshake includes an “Upgrade” header, indicating the desire to establish a WebSocket connection.

**Full-Duplex Communication:**

* WebSockets provide full-duplex communication, allowing both the client and server to send messages independently at any time. This bidirectional communication eliminates the need for multiple HTTP requests for updates.

**Persistent Connection:**

* Once the WebSocket connection is established, it remains open for the duration of the communication session. This persistence eliminates the overhead of opening and closing connections for each message, contributing to lower latency.

**Frame-based Communication:**

* WebSocket messages are transmitted in frames. Frames can be either text or binary, and they include data payloads. This frame-based structure allows for efficient and organized data transfer.

**Low Latency and Real-time Updates:**

* WebSockets significantly reduce latency compared to traditional polling methods. Real-time updates are achieved as messages are sent and received promptly over the established WebSocket connection, making them suitable for applications requiring instant communication, such as chat applications or live data streaming.

**Use Cases:**

**Real-time Chat Applications:**

* WebSockets enable instant message delivery in chat applications, providing a responsive and seamless user experience without the need for constant server polling.

**Online Gaming:**

* In gaming, WebSockets facilitate low-latency communication between players and servers, ensuring timely updates on game states, player actions, and multiplayer interactions.

**Financial Applications:**

* WebSockets are crucial in financial platforms for real-time market data streaming, allowing traders to receive instantaneous updates on stock prices and market changes for informed decision-making.

**Explain the WebSocket API and how it is used in both server and client applications.**

The WebSocket API provides a standardized interface for establishing and managing WebSocket connections in both server and client applications. In the client, it typically involves creating a WebSocket instance, handling events like onopen and onmessage, and sending/receiving messages. On the server side, developers use the API to handle incoming connections, manage communication events, and send messages to connected clients. This API simplifies the implementation of real-time, bidirectional communication between web applications and servers.

**Installation**

To get started, you’ll need to install the websockets library. You can easily install it via pip:

pip install websockets

**Creating a WebSocket Server**

Let’s create a simple WebSocket server using websockets.

import asyncio  
import websockets  
  
async def hello(websocket):  
 name = await websocket.recv()  
 print(f'Server Received: {name}')  
 greeting = f"Hello {name}!"  
  
 await websocket.send(greeting)  
 print(f'Server Sent: {greeting}')  
  
async def main():  
 async with websockets.serve(hello, "localhost", 8765):  
 await asyncio.Future() # run forever  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 asyncio.run(main())

**Creating a WebSocket Client**

Now, let’s create a WebSocket client to communicate with the server:

import websockets  
import asyncio  
  
async def hello():  
 uri = "ws://localhost:8765"  
 async with websockets.connect(uri) as websocket:  
 name = input("What's your name? ")  
  
 await websocket.send(name)  
 print(f'Client sent: {name}')  
  
 greeting = await websocket.recv()  
 print(f"Client received: {greeting}")  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 asyncio.run(hello())



**Why Use**websockets**?**

The websockets library is designed with four principles in mind:

* Correctness: It’s heavily tested for compliance with RFC 6455.
* Simplicity: All you need to understand is msg = await ws.recv() and await ws.send(msg). The library handles managing connections so you can focus on your application.
* Robustness: It’s built for production. For example, it was the only library to handle backpressure correctly before the issue became widely known in the Python community.
* Performance: Memory usage is optimized and configurable. A C extension accelerates expensive operations. It’s pre-compiled for Linux, macOS, and Windows and packaged in the wheel format for each system and Python version.

However, if you prefer callbacks over coroutines or if you’re looking for a mixed HTTP/WebSocket library, you might want to pick another library. If you want to do both on the same server, look at HTTP frameworks that build on top of WebSockets to support WebSocket connections, like Sanic.

**Handling Events and Actions**

You can extend the server and client logic to handle different events and actions, such as message parsing, broadcasting messages to multiple clients, handling errors, and more, by building on top of the basic send-receive functionality demonstrated above.

**Conclusion**

The websockets library in Python simplifies working with WebSockets, enabling real-time communication between clients and servers. By using this library, developers can implement WebSocket-based applications for various purposes, including chat applications, real-time data streaming, and more.

Experimenting with the provided examples and exploring the library’s documentation will help in understanding the full potential of websockets in Python. For more such insightful contents, do visit my profile and checkout other articles. Happy Learning !!!

**Stackademic**

*Thank you for reading until the end. Before you go:*

* *Please consider****clapping****and****following****the writer! 👏*
* *Follow us on*[***Twitter(X)***](https://twitter.com/stackademichq)*,*[***LinkedIn***](https://www.linkedin.com/company/stackademic)*, and*[***YouTube***](https://www.youtube.com/c/stackademic)***.***
* *Visit*[***Stackademic.com***](http://stackademic.com/)*to find out more about how we are democratizing free programming education around the world.*