

Amartya Choudhury

JU-BCSE IV

Roll – 001610501026

Internet Technology Lab

Key value store with websockets

A python based in-memory key value store using websockets and asyncio

Problem Statement:

Implement a key-value store using Websocket. The server implements the key-value store and clients make use of it. The server must accept clients’ connections and serve their requests for ‘get’ and ‘put’ key value pairs. All key-value pairs should be stored by the server only in memory. Keys and values are strings as in Assignment 1. Implement authorization so that only few clients having the role “manager” can access other’s key-value stores. A user is assigned the “guest” role by default. The server can upgrade a “guest” user to a “manager” user.

WebSocket:

Long Polling

It is a technology where the client requests information from the server without expecting an immediate response or basically involves making an HTTP request to a server and then holding the connection open to allow the server to respond later. Using long polling the server allows approximately 6 parallel connections from the browser.

Load balancing in this is easy compared to other ways. Long polling is the oldest ways and hence is supported on all web browsers. Though due to the fewer updates in this it does not provide re-connection handling. Long polling is a lot more intensive or heavy on the server, but more widely accepted for browsers.

WebSockets

WebSocket is a computer communication protocol that enables us full-duplex communication channels over a single transfer control protocol (TCP) connection. The WebSocket protocol enables interaction between a web browser and a web server with low weight overheads, providing real-time data transfer from and to the server. This is done by defining a standard way for the server to send information to the client without being first requested by the client, and then allowing messages to be passed to and fro while keeping the connection open. In this way, a two-way advancing conversation can take place between the client and the server without any issue.

Websockets are majorly accepted in web browsers such as google chrome, opera, edge, firefox, safari etc. WebSockets is light on the browser and it provides up to 1024 parallel connections from the browser. It has a complicated load balancing and proxying technique. It also supports dropped client detection which was absent in long polling but it also does not provide re-connection handling.

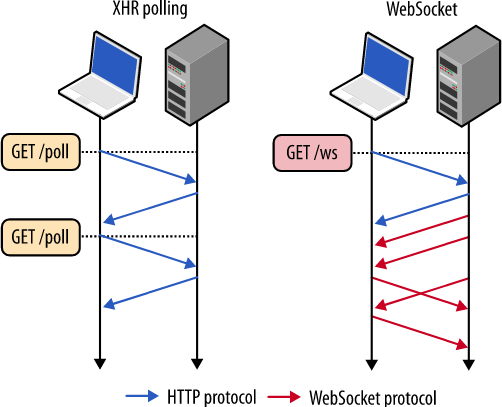


Fig 1: Long Polling vs Websocket

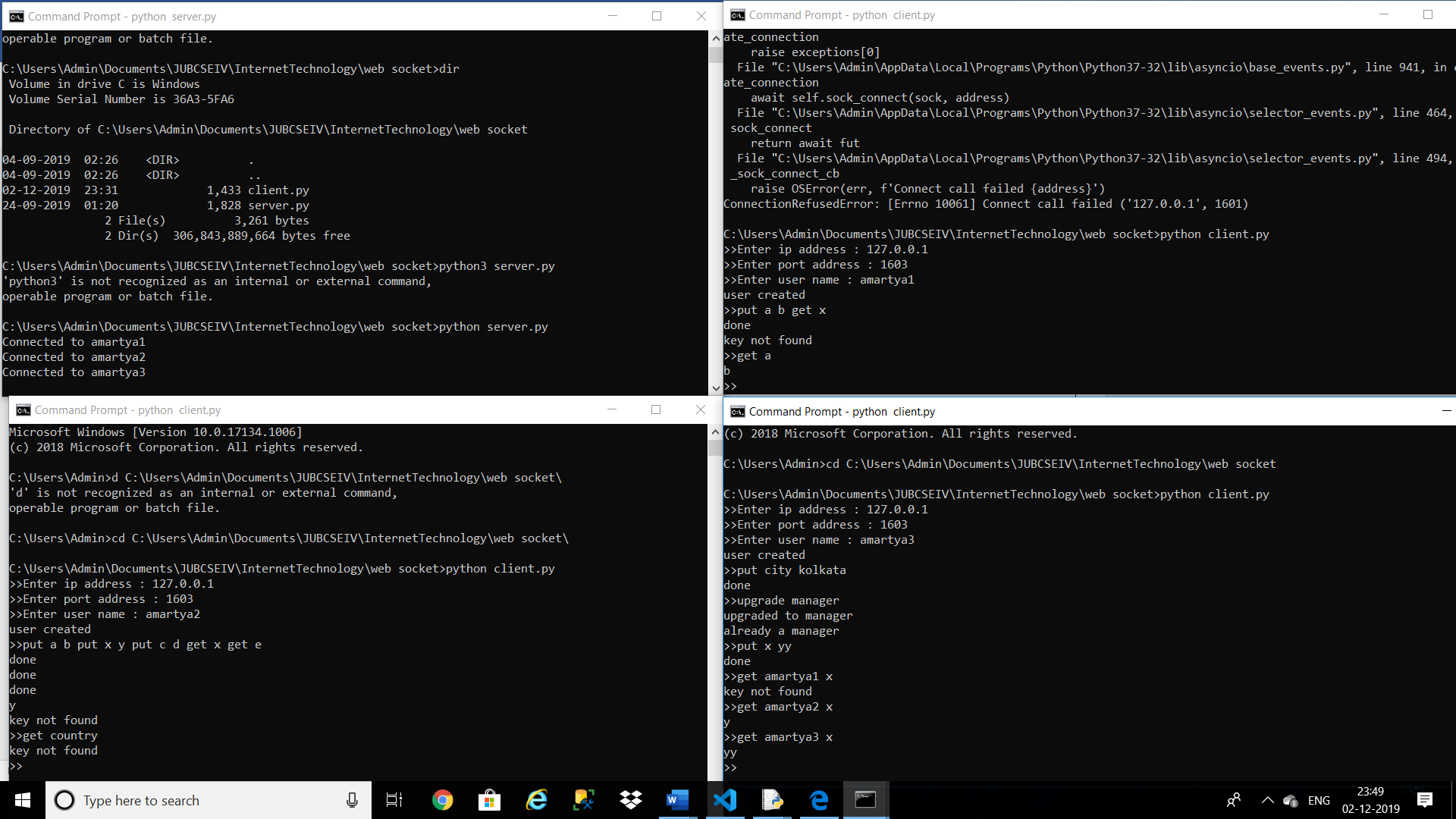
Client.py

|  |
| --- |
| import asyncio  import websockets  addr = str(input('>>Enter ip address : '))  port = int(input('>>Enter port address : '))  cmds = ['get','put','upgrade','q']  num\_cmd = 4  async def connect():  uri = "ws://"+addr+":"+str(port)  async with websockets.connect(uri) as websocket:  while True:  if not websocket.open:  print("Websocket not open. Trying to reconnect")  websocket = await websockets.connect(uri)  user = str(input('>>Enter user name : '))  await websocket.send(user)  ans = await websocket.recv()  if ans == 'success':  print('user created')  break  else:  print(str(ans))  while True:  if not websocket.open:  print("Websocket not open. Trying to reconnect")  websocket = await websockets.connect(uri)  x = str(input(">>"))  words = x.split()  for i in range(len(words)):  if words[i] in cmds:  x=words[i]  if x == 'upgrade' or x=='q':  await websocket.send(x)  x2 = await websocket.recv()  print(str(x2))  if x=='q':  break  elif (i==len(words)-1) or (words[i+1] in cmds):  x=x+' '+words[i]  boo=False  for j in range(num\_cmd):  boo = boo or x.startswith(cmds[j])  if boo:  await websocket.send(x)  x2 = await websocket.recv()  print(str(x2))  else:  print('wrong message format')  else:  x=x+' '+words[i]  if x=='q':  break  asyncio.get\_event\_loop().run\_until\_complete(connect()) |

Server.py

|  |
| --- |
| **import** **asyncio**  **import** **websockets**  **class** **Client**:  **def** **\_\_init\_\_**(self, a):  self.name = a  self.dct = {'role':'guest'}  clients = {}  async **def** **hello**(websocket, path):  **while** **True**:  x = await websocket.recv()  **if** x **in** clients:  await websocket.send('duplicate user name')  **else**:  await websocket.send('success')  print('Connected to '+x)  **break**  c = Client(x)  clients[x] = c  # async for data in websocket:  **while** **True**:  data = await websocket.recv()  arr = data.split()    **if** arr[**0**] == 'q':  await websocket.send('exiting')  **break**    **elif** arr[**0**] == 'put':  **if** len(arr) == **3**:  c.dct[arr[**1**]] = arr[**2**]  await websocket.send('done')  **else**:  await websocket.send('wrong message format')  **elif** arr[**0**] == 'get':  **if** len(arr) == **2**:  **if** arr[**1**] **in** c.dct:  await websocket.send(c.dct[arr[**1**]])  **else**:  await websocket.send('key not found')  **elif** len(arr) == **3**:  **if** c.dct['role'] == 'guest':  await websocket.send('you are not allowed to access keys of other users'.encode('utf-8'))  **elif** arr[**1**] **in** clients:  c2 = clients[arr[**1**]]  **if** arr[**2**] **in** c2.dct:  await websocket.send(c2.dct[arr[**2**]])  **else**:  await websocket.send('key not found')  **else**:  await websocket.send('user not found')  **else**:  await websocket.send('wrong message format')  **elif** arr[**0**] == 'upgrade':  **if** c.dct['role'] == 'guest':  c.dct['role'] = 'manager'  await websocket.send('upgraded to manager')  **else**:  await websocket.send('already a manager')  **else**:  await websocket.send('wrong message format')  **del** clients[c.name]  addr = "127.0.0.1"  port = **1603**  start\_server = websockets.server.serve(hello, addr, port, ping\_interval = **1000**, ping\_timeout = **1000**)  asyncio.get\_event\_loop().run\_until\_complete(start\_server)  asyncio.get\_event\_loop().run\_forever() |

Output:



Conclusion:

Web Sockets enable the server and client to send messages to each other at any time, after a connection is established, without an explicit request by one or the other. In a real time systems , clients do not know when fresh data is available, with web sockets the server can push new data at any time to the clients.

References :

1. <https://www.tutorialspoint.com/websockets/websockets_overview.htm>
2. https://websockets.readthedocs.io/en/stable/intro.html