# Websocket server on amazon ec2

### What are WebSockets?

A WebSocket is a bi-directional connection between a client and a server which means that both the client and the server can stream messages to each other independently. A WebSocket uses HTTP as the initial communication mechanism, the Transmission Control Protocol (TCP) connection will remain open so that it can be used to send Realtime messages between the client and server.  
  
Unlike using a standard HTTP connection where the server can only send a message to the client when a request is made, a WebSocket connection allows the server to push data in real time to the client without waiting for a request.



## HTTP

## Websocket

Client

Client

Server

Server

Request

Response

Request

Response

Request handshake

Accept handshake

Realtime bi-directional messaging

Request to close connection

Closing connection

### Setting up Amazon EC2 instance

An Amazon EC2 instance provides a cloud based Virtual Machine (VM), this VM can be used to host servers. In this case it will be used to host a WebSocket server.

##### Pricing.

The costs of running the EC2 instance depends on the computing power needed for the application. The price of the selected instance type will be charged every hour the instance is running, so stopping the instance when not in use will reduce costs significantly.  
More details of the pricing for the computing power can be found here [1].  
  
Next to computing a VM also needs storage, for that purpose Amazon offers Elastic Block Storage (EBS). When setting up a new EC2 instance an EBS volume has to be added. This EBS volume must be bigger than the Machine Image used. The price of EBS volumes is starting at $0,10 per GB/month depending on the region. Be aware that the costs are calculated over the total size of the volume, not just the storage that is in use. The costs will also accumulate when the instance that it was connected to is stopped or terminated.  
More details of the pricing for EBS volumes can be found here [2].

##### Free tier.

Amazon offers a generous free tier for new users. The first 12 months after registering a new account for Amazon Web Services (AWS) you are eligible to try some services for free, this includes everything needed to set up a fully functional WebSocket server.   
These services are:  
- up to 30 GB/month of free EBS  
- one free t2.micro instance that has one 2.5 GHz CPU and 1 GiB (Gibibyte) memory.  
More details about the free tier can be found here [3].

##### Creating the EC2 instance.

The first thing that is needed is an AWS account. After creating an account an EC2 instance could be launched. When launching the instance a few things can be customized.  
After you are logged in the instance can be launched here [4]. There are seven steps required to create the EC2 instance.

##### 1. Choose an Amazon Machine Image (AMI).

In this step you can select your preferred software configuration like an operating system. This is required to launch any instance. Be aware that for some AMI’s there is no free tier available, be sure to check if the preferred AMI is free tier eligible.   
  




For the rest of this document we assume that the instance is running the newest version of Amazon Linux 2.

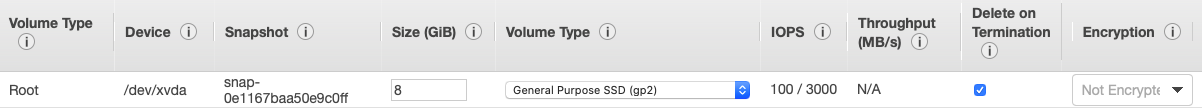
##### 2. Choose an Instance Type.

Amazon provides a selection of different combinations of TCP, memory, storage and network capacity, select one that offers appropriate resources for your application. T2.micro is available within the 12 month free tier, prices of the other types are found here [1].

##### 3. Configure Instance Details.

Here we can configure a variety of different settings like the number of instances and management roles. This can be left as is when setting up a single instance used to host our WebSocket Server.

##### 4. Add Storage.

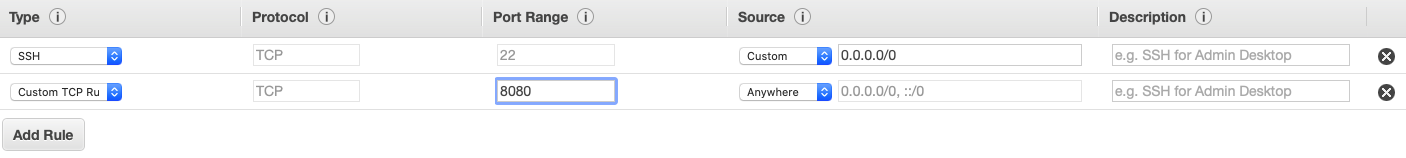
The instance will be launched with the storage devices that are set up here. You can attach multiple storage devices to the instance, or only use the root volume. Also, the size of the root volume and optional extra volumes can be customized here. The 12-month free tier offers 30 GB of General Purpose SSD (gp2) for free. I recommend checking the Delete on Termination checkbox, this will delete the volume when the instance that is it connected to is terminated. This will prevent costs for a storage volume that is not linked to any instance.  




##### 5. Add Tags.

We will not be using tags for our WebSocket Server, so we will leave this as is.  
More information about tags can be found here [5].

##### 6. Configure Security Groups.

Here we can specify rules about traffic to and from our instance.  
The top line is set by default, this allows port 22 to be open for a Secure Shell (SSH) connection with the server administrator(s). This connection can only be established if the person trying to connect has access to a key file that is generated when the instance is launched.  


To allow any client to connect to our server we will have to open another port for that.  
A port(range) can be opened by adding a new Custom TCP Rule, specify the open port(range) any allow connections from Anywhere.

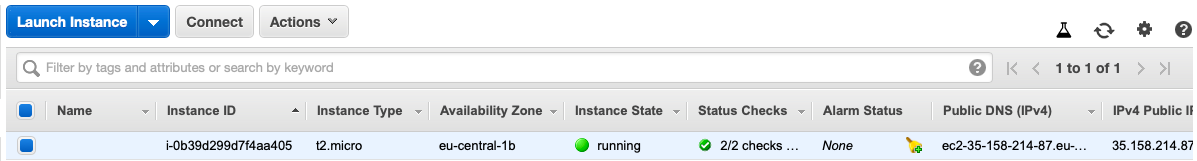
##### 7. Review.

Review if everything is set up correctly and launch the instance.

##### Key Pairs.

A key pair is a file that stores the private key that allows you to securely SSH into your instance. You can create a new one here or use an existing one. This key has to be downloaded before the instance can be launched. This key is like a password to access the server, if this file is lost the server can never be accessed again. And everyone who has access to the file can access the server. So be sure to back this up properly and safe.

##### Connect to the server as administrator.

To make any modifications to the server the administrator can connect to the server using a SSH connection. Select the instance you want to connect to, then click connect. The steps to connect will be provided there.  




### Writing the software.

For this example we will be creating a simple WebSocket server that forwards messages received by Cient1 to Client2. The advantage of using WebSockets over HTTP in this application is that Client2 does not have to make requests to ask if there are massages. As soon as Client1 sends a message to the Server Client 2 will get the message from the server.



Client1



Server



Client2

Request handshake

Accept handshake

Request handshake

Accept handshake

Identify as sending side

Identify as receiving side

Message Stream

Message Stream

Request to close connection

Closing connection

Request to close connection

Closing connection

The code for the Clients and the Server can be written in different languages e.g. Python, C, C++, Java and NodeJS. They all offer different pros and cons, for this example we will use NodeJS for both the Client side and the Server side. We will use the ws (WebSocket) package to handle the WebSocket connection.  
NodeJS documentation can be found here [6].  
ws documentation can be found here [7].

##### Server-side.

The first step for programming the Server is to connect as administrator using an SSH connection. Create a project folder in the root directory that will contain our Server-side code.

Install NodeJS, follow instructions found here [8]. Also install the ws package in the project folder npm install ws.

Now use Vim, this is a text editor that is already preinstalled on some versions of Linux including Amazon Linux to create and open a server.js file. This can be done with this command vim server.js

Paste the code from the server.js file in this git repository [9] into the server.js file. Explanation about the code is in the .js file as comments. To close and save press ESC and type :wq

The code can be ran in two different ways.

The most simple way to run the server side code is node server.js . This is only recommended for debugging because it only runs as long as the SSH connection is active.

To keep the server active when the SSH connection is disconnected you have to use a different screen than the default.   
To create or reconnect to a screen use screen -R screen-name . Run the server-side code in this screen (node server.js).

##### Client-side.

Also create a project folder and install NodeJS and the ws package in it.   
There are some great tutorials on how to install NodeJS and npm (node package manager) on different operating systems online.

- [Windows](https://www.guru99.com/download-install-node-js.html)  
- [Linux](https://www.devroom.io/2011/10/24/installing-node-js-and-npm-on-ubuntu-debian/)  
- [Mac](https://treehouse.github.io/installation-guides/mac/node-mac.html)

Use npm to install the ws package npm install ws.

Download the transmitter and receiver .js files from this git repository [9] and run them in two different windows. You should see the message that is sent by the transmitter to the server appear in the receiving client’s log.

### Multi-client support

To support more than just one client for one function you can replace the transmitter and receiver variables with an array to store multiple values. Append the newly identified client to the correct array. When the servers receives a message from a transmitting client it should loop through the receiving client array and forward the message to all receiving clients.

Theoretically there can be an infinite amount of transmitters and receivers.  
These changes are made in the multi-client-server.js found in my GitHub [9].  
  
Version history



Transmitter 1



Transmitter 2



Receiver 2



Receiver 1



Server

31-5-2020 - 1.0 Sander Benten - Initial version.  
16-6-2020 - 1.1 Sander Benten - Added multi-client support + slight improvements.

### References.

[1] <https://aws.amazon.com/ec2/pricing/on-demand/>

[2] <https://aws.amazon.com/ebs/pricing/>

[3] <https://aws.amazon.com/free>

[4] <https://console.aws.amazon.com/ec2/>

[5] <https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/Using_Tags.html>

[6] <https://nodejs.org/en/docs/>

[7] <https://www.npmjs.com/package/ws>

[8] <https://docs.aws.amazon.com/sdk-for-javascript/v2/developer-guide/setting-up-node-on-ec2-instance.html>

[9] <https://github.com/Sander-Benten/EC2-websockets>