

FishNet WebGL WSS/SSL Example Guide

Table of Contents

Intro	2
Namecheap	2
DigitalOcean	3
File transfer	5
Unity Scripting	5
Build and Deploy Server	14
Starting the server	15
Conclusion	20

Intro

The goal of this guide is to help you connect to a FishNet server hosted on a DigitalOcean droplet from a WebGL client using WSS/SSL so that you may host your game on a secure website like itch.io. However, non-secure HTTP communication will also work. For the extent of this guide, I will use the example website "example.xyz". I personally do not have much knowledge of websites and websockets (and none at all before doing this), so forgive me if I don't explain things well enough.

Namecheap

First, you need to buy a website domain. You can use <https://www.namecheap.com/> or whichever site you prefer. I bought "example.xyz".

After purchasing the domain, (in Namecheap) you need to head into the Dashboard/Domain List and click the "Manage" button on the right side of the domain you want to use. Scroll down to the "NAMESERVERS" section and from the dropdown menu select "Custom DNS".

Now, add three nameservers:

1. ns1.digitalocean.com
2. ns2.digitalocean.com
3. ns3.digitalocean.com

This will connect your domain with your droplet.

Note that your website may take time (30min-1hr) to activate or establish connection with DigitalOcean servers.

NAMESERVERS

?

Custom DNS ▼

ns1.digitalocean.com

ns2.digitalocean.com

ns3.digitalocean.com

+ ADD NAMESERVER

DigitalOcean

You need to register a DigitalOcean account, create your first project (should give you a default one), and purchase a droplet.

At the time of writing this guide, I chose the Ubuntu 22.04 x64 image, basic plan with the cheapest option. Optionally and preferably, you should use SSH authentication.

Go into your project, and click on the ellipsis menu for your droplet on the right side of the droplets list and choose "Add a domain". Type in the domain you just bought (example.xyz), set the project, and click "Add Domain".

Once added, you can click on your domain to create new records. You will see three records already added, which is why we added the namespaces previously when you bought your domain. We will need to add two more A records: "@" (stands for apex domain) and "www". Make sure the "A" tab is selected and type each of these in as the host name, select your droplet from the dropdown menu, (optionally change the TTL), and press "Create Record".

DNS records

Type	Hostname	Value	TTL (seconds)	
A	www.gooby.xyz	directs to 137.184.71.96	3600	More ∨
A	gooby.xyz	directs to 137.184.71.96	3600	More ∨
NS	gooby.xyz	directs to ns1.digitalocean.com.	1800	More ∨
NS	gooby.xyz	directs to ns2.digitalocean.com.	1800	More ∨
NS	gooby.xyz	directs to ns3.digitalocean.com.	1800	More ∨

Next, you need to set up your server. Here are several guides you should follow in order and *thoroughly*. If needed, select your current server version for the guide at the top. Ensure there are no errors along the way. If there are, try again or ask for help in Discord.

1. <https://www.digitalocean.com/community/tutorials/initial-server-setup-with-ubuntu-22-04>
 - You don't need to add users now, you can just do step 4 and continue to the next guide
2. <https://www.digitalocean.com/community/tutorials/how-to-install-nginx-on-ubuntu-22-04>
 - At the end of step 5, make sure you type "http://example.xyz" instead of your IP to get the correct page.
3. <https://www.digitalocean.com/community/tutorials/how-to-secure-nginx-with-let-s-encrypt-on-ubuntu-22-04>
 - You may skip the prerequisites section because we already did all of them!
 - You may also skip step 2 since that was already done in the previous guide.

Once all the tutorials are complete with no errors, we need to generate an actual cert.pfx file that the server can read.

In the Linux terminal, navigate to your domains letsencrypt directory by typing:

```
cd /etc/letsencrypt/live/example.xyz/
```

And then put:

```
openssl pkcs12 -export -out cert.pfx -inkey privkey.pem -in cert.pem  
-certfile chain.pem -leg
```

Assign a password if you want or just leave it blank. This will generate a cert.pfx file compatible with Unity's version of Mono and our built server.

File transfer

Now, we need to put the file in our server folder, which we haven't made yet.

In order to transfer files to our server, you have two options:

- a. If you're familiar with Linux, you can use the terminal to transfer files over, or
- b. Follow this guide to install and set up FileZilla to have a more visual file transfer if you don't have much experience:

<https://docs.digitalocean.com/products/droplets/how-to/transfer-files/>

Unity Scripting

We also need to program and build a server in Unity and script an echo server example for testing.

Make sure your Unity version includes the modules for "Linux Dedicated Server Build Support" and "WebGL Build Support". You can check these in Unity Hub.

Linux Dedicated Server Build Support	Installed	411.38 MB
<input type="checkbox"/> Mac Build Support (Mono)	319.74 MB	1.75 GB
<input type="checkbox"/> Mac Dedicated Server Build Support	317.95 MB	1.73 GB
<input type="checkbox"/> Universal Windows Platform Build Support	276.79 MB	1.94 GB
WebGL Build Support	Installed	1.62 GB

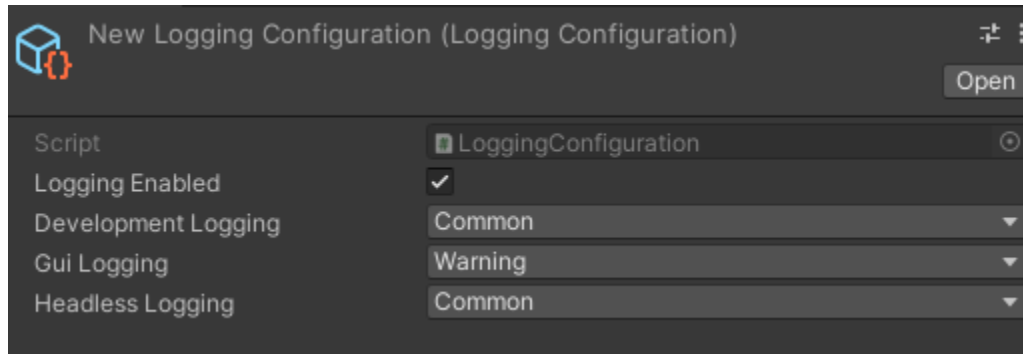
Make a new project (2021.5 LTS at the time of writing) and import the latest versions of:

1. [FishNet](#) and
2. [Bayou](#) (FishNet's web socket transport)

If you want to skip this scripting section, you may install this Unity package I made for the example found here: <https://github.com/celojevic/FishNet-WebGL-WSS-SSL-Example>

First, **set up the NetworkManager** by searching for the prefab included with FishNet and dragging it into the scene. Optionally, you may unpack the prefab. You may remove the ObserverManager script from it as it may cause issues since we aren't necessarily setting it up correctly in this tutorial.

Also, you should **create a new Logging config** by right clicking your project > Create > FishNet > Logging > Logging Config. Set the "Headless Logging" to "common". Assign this object to the NetworkManager component. This will help see what's going on in the server later when we start it.







Add these components to the NetworkManager:

- **TransportManager**
- **Multipass**, and assign it to the TransportManager's Transport field
- **Tugboat**, and add it to Multipass' Transports list
- **Bayou**, and add it to Multipass' Transports list

Tugboat will be used to test a PC build or editor connection to the server. Set its client address to your **droplet's IP** (optionally, for local testing leave it as "localhost").

Bayou will be used to communicate with the server from a WebGL build.

- Enable "Use Wss".
- Expand the "SSL Configuration" foldout. Enable "Enabled". Set the certificate path to `./cert.pfx` since it will be in the same directory level as the server executable. Set the "Certificate Password" to be the password you set above when you generated the certificate.
- Change Bayou's port to 7771 or any other port that doesn't match Tugboat's port. You cannot open two servers to listen on the same port.
- Change the client address to your **website domain name (example.xyz)**, not your droplet IP. This is because we used NGINX to reroute communications via our domain to our droplet instead of communicating directly with the droplet like Tugboat would.





▼  Transport Manager (Script)   

Script

TransportManager

Transport

NetworkManager (1) (Multipass)

▼  Multipass (Script)   

Script

Multipass

Global Server Actions

☒

▼ Transports

2

Element 0





NetworkManager (1) (Tugboat)

Element 1

NetworkManager (1) (Bayou)

+

-

▼  Tugboat (Script)   

Script

Tugboat

Channels

Unreliable MTU

1023

Server

Attack Response Type

Warn And Kick

Ipv 4 Bind Address

Ipv 6 Bind Address

Port

7770

Maximum Clients

4095

Client





Client Address

128.0.0.1

Misc

Timeout

15

▼  Bayou (Script)   

Script

Bayou

Security

Use Wss

☒

▼ Ssl Configuration

Enabled

☒

Certificate Path

./cert.pfx

Certificate Password

Ssl Protocol

None

Channels

Mtu

1023

Server

Port

7771

Maximum Clients

2000

Client

Client Address

localhost

Echo Server Test

We need to make some scripts for the echo server.

First, a *TransportSetter*. Attach it to the NetworkManager. This just sets the client transport based on the build. So, WebGL clients will use Bayou, and everything else will use Tugboat.

```
using FishNet.Transporting.Bayou;
using FishNet.Transporting.Multipass;
using FishNet.Transporting.Tugboat;
using UnityEngine;

public class TransportSetter : MonoBehaviour
{
    private void Awake()
    {
        Multipass mp = GetComponent<Multipass>();

#if UNITY_WEBGL && !UNITY_SERVER && !UNITY_EDITOR
        mp.SetClientTransport<Bayou>();
#else
        mp.SetClientTransport<Tugboat>();
#endif

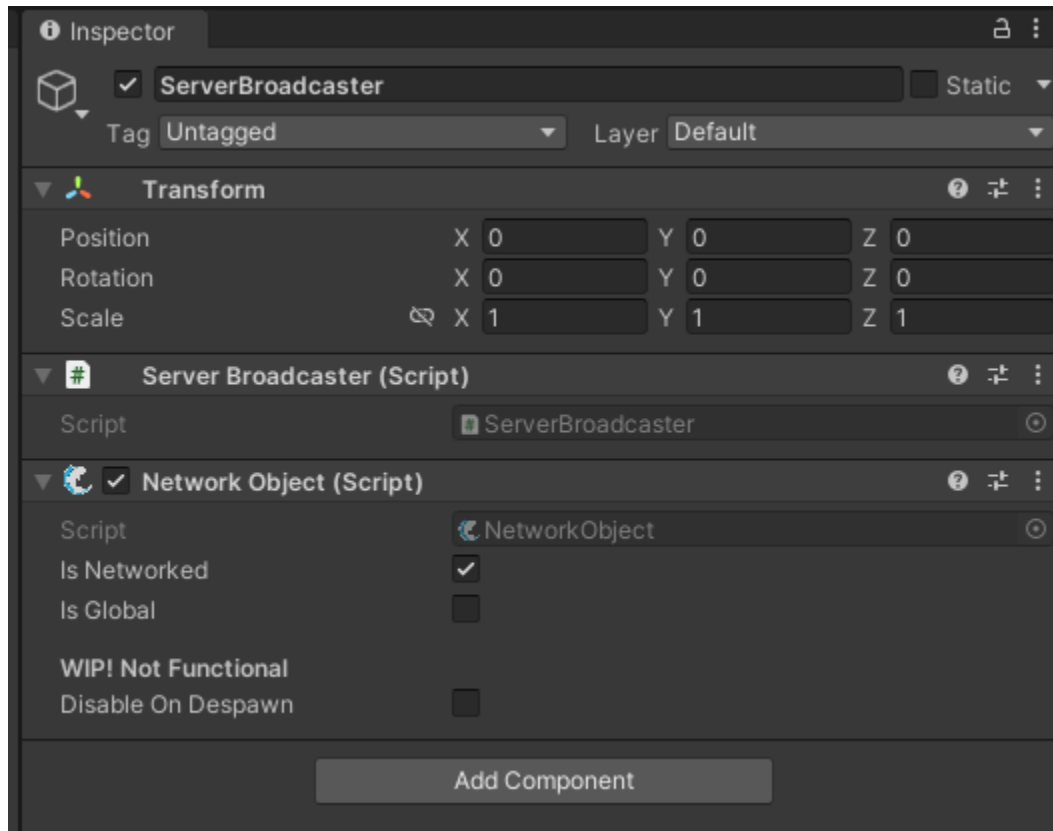
        Debug.Log("Client transport set as " + mp.ClientTransport);
    }
}
```

Then, a simple broadcast struct named *ChatMsg* we will use to send strings to/from the server.

```
using FishNet.Broadcast;

public struct ChatMsg : IBroadcast
{
    public string Text;
}
```


Next, a *ServerBroadcaster* that will register the ChatMsg broadcast on the server and handle it by appending the client the ID to the message and broadcasting it back to all connected clients. Put this script on a new empty object in your scene. It should automatically attach a *NetworkObject* component with it as well.



```

using FishNet.Connection;
using FishNet.Object;
using UnityEngine;

public class ServerBroadcaster : NetworkBehaviour
{
    public override void OnStartServer()
    {
        base.OnStartServer();

        ServerManager.RegisterBroadcast<ChatMsg>(OnChatMsg);
    }

    public override void OnStopServer()
    {
        base.OnStopServer();

        ServerManager.UnregisterBroadcast<ChatMsg>(OnChatMsg);
    }

    private void OnChatMsg(NetworkConnection conn, ChatMsg msg)
    {
        Debug.Log($"Client {conn.ClientId} sent msg: {msg.Text}");

        msg.Text = $"{conn.ClientId}: {msg.Text}";

        ServerManager.Broadcast(msg);
    }
}

```

Finally, the actual echo canvas UI script named *EchoCanvas*. This will register the ChatMsg broadcast on the client so when we receive a message, it'll add the text to a TMP text object. It will also send ChatMsg's to the server via the input field's OnEndEdit callback.

```

using FishNet.Object;
using TMPro;
using UnityEngine;

public class EchoCanvas : NetworkBehaviour
{
    [SerializeField] private TMP_Text _text = null;
    [SerializeField] private TMP_InputField _input = null;

    private void Awake()
    {
        _text.text = "";

        _input.onEndEdit.AddListener((text) =>
        {
            if (!ClientManager.Started) return;
            if (string.IsNullOrEmpty(_input.text)) return;

            ClientManager.Broadcast(new ChatMsg
            {
                Text = _input.text
            });
            _input.text = "";
        });
    }

    public override void OnStartClient()
    {
        base.OnStartClient();
        ClientManager.RegisterBroadcast<ChatMsg>(OnChatMsg);
    }

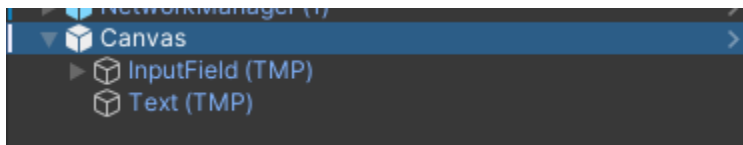
    public override void OnStopClient()
    {
        base.OnStopClient();
        ClientManager.UnregisterBroadcast<ChatMsg>(OnChatMsg);
    }

    private void OnChatMsg(ChatMsg msg)
    {
        _text.text += msg.Text + "\n";
    }
}

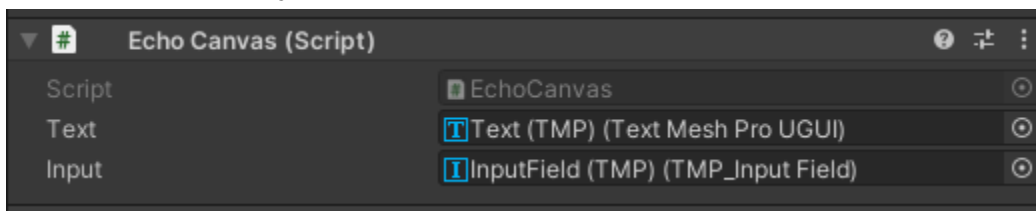
```

You need to make the actual UI now.

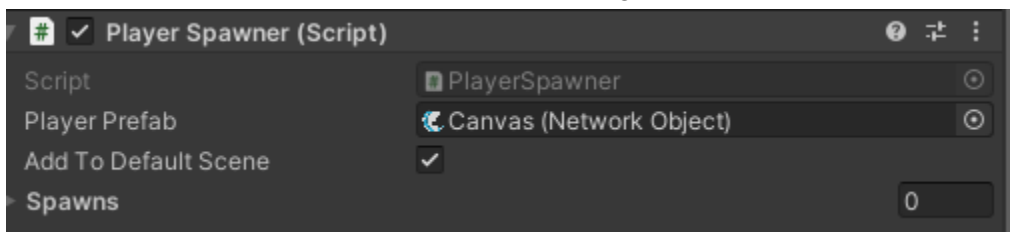
- Add a canvas.
- Add a TMP input field as a child of the canvas (import TMP if needed). Stretch it along the bottom of the screen.
- Add a TMP text as a child of the canvas as well. Stretch it across the entire screen. Adjust the top/bottom positions so it is not covered by the NetworkHudCanvas buttons and the input field. Align it to the bottom left in the TMP component.



- Add the EchoCanvas component we just made to the root canvas object, and assign the input field and text objects to it.

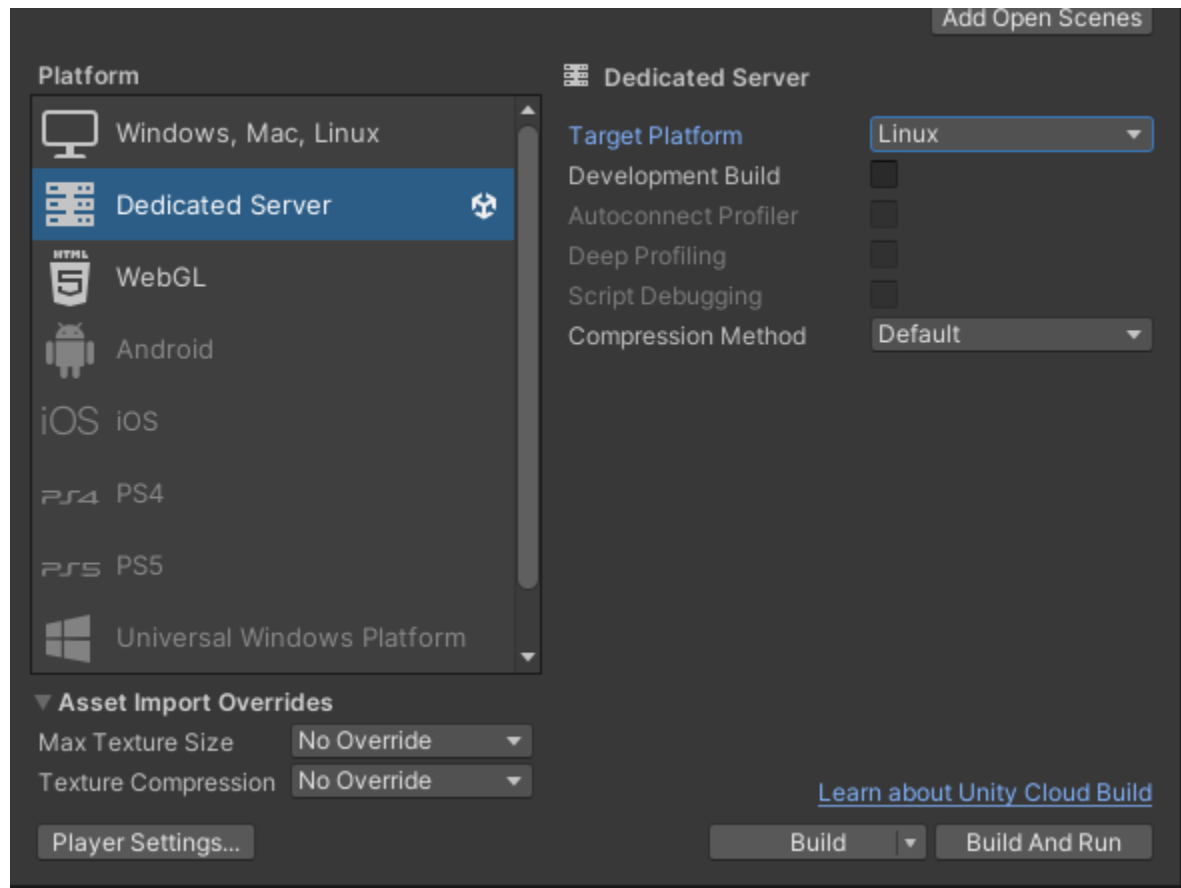


- This should automatically add the NetworkObject component as well. Drag the canvas into your project to make it a prefab, and assign it to the PlayerSpawner component in the appropriate field that's on the NetworkManager.



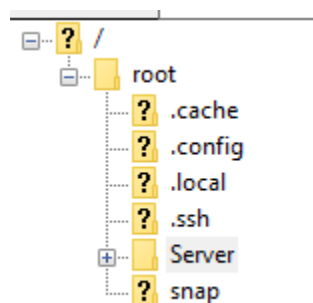
Build and Deploy Server

Now, we can build the server by going to *File > Build settings*, and switch the platform to "Dedicated server" and its "Target Platform" to "Linux".



Then press "Build" and choose a directory to build to, ideally a "Builds/Server" folder in the project root directory to keep things separated.

Once done building, open FileZilla or a terminal, however you are transferring files over, and transfer over the entire *Server* folder over to the "root" directory on the droplet. The hierarchy on FileZilla would look like this:

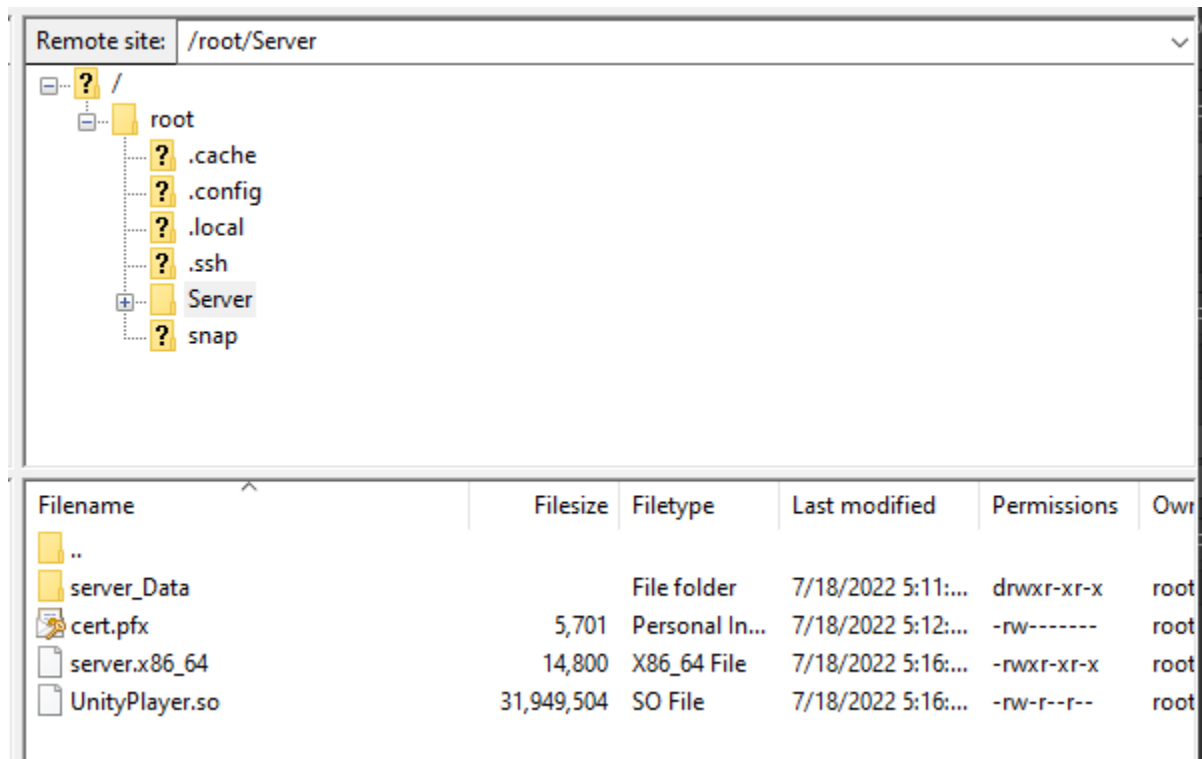


Starting the server

Go back to the droplet console. You need to copy over the cert.pfx we generated previously into the Server folder you just transferred over by typing in the terminal (Don't forget the tilde~ and to replace *example.xyz* with your domain name):

```
cp /etc/letsencrypt/live/example.xyz/cert.pfx ~/Server/cert.pfx
```

Change the server directory to match yours (/Server/ in my case, **NOTE** that it's all **case-sensitive**), but the cert.pfx should be in the same directory level as the server executable.



Run the server by first navigating to your server directory by typing:

```
cd ~/Server/
```

NOTE: This is **very important** since Linux will use this as the relative path for everything, including finding the cert.pfx file. So, make sure every time you start your server, you cd this directory, or adjust the certificate path in the Bayou component.

Then, mark the server as executable:

```
chmod +x ./server.x86_64
```

And then running the server (this command does not run it continuously when you close the console nor in the background, so you can use Ctrl+C to stop it or restart the console):

```
./server.x86_64
```

Continuous command:

```
nohup ./server.x86_64
```

Continuous and run in background command:

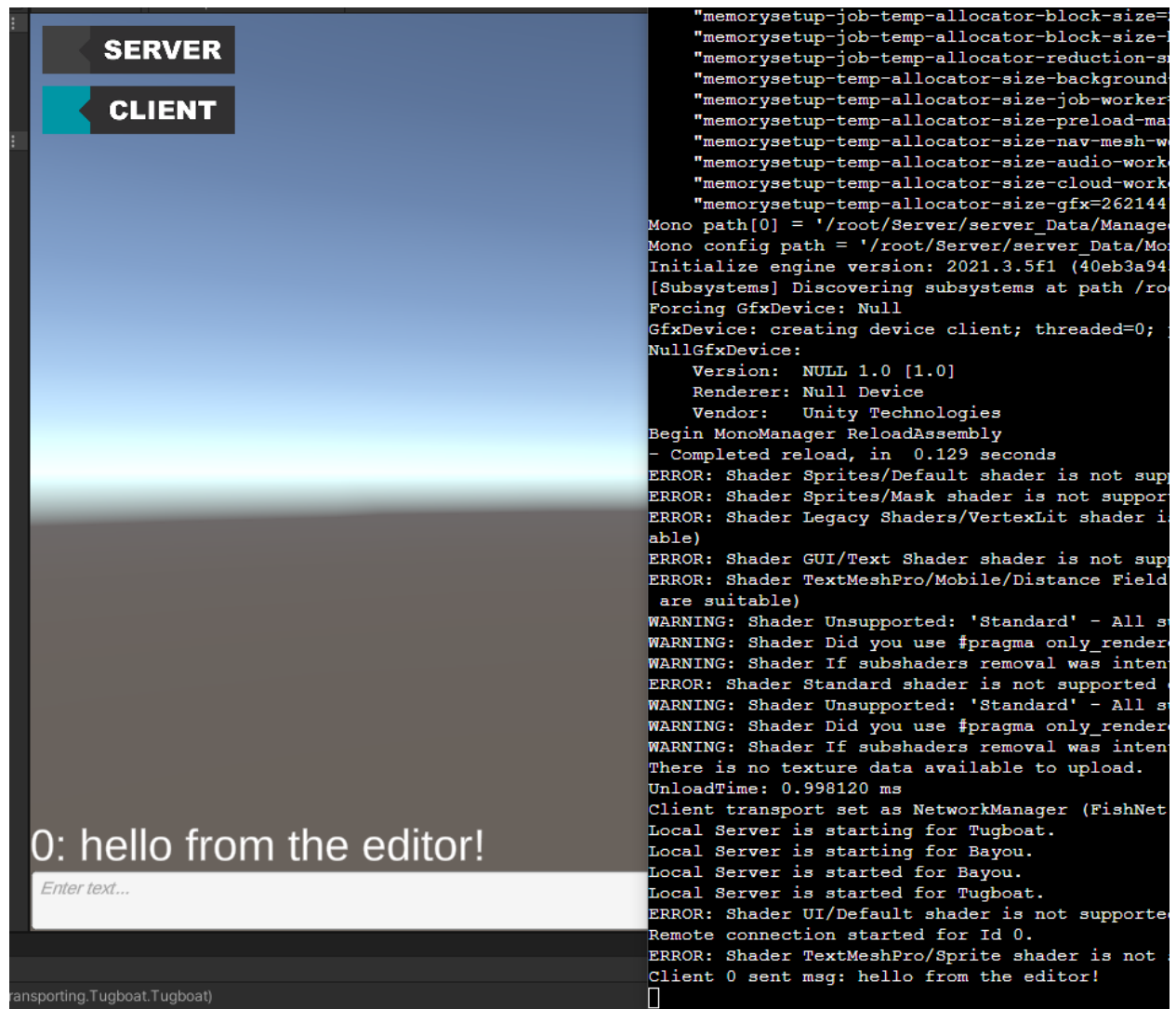
```
nohup ./server.x86_64 &
```

If you see a lot of shader warnings and errors, ignore them, it's totally normal for Unity and won't affect the server.

In the console you should see some success messages:

```
Client transport set as NetworkManager
Local Server is starting for Tugboat.
Local Server is starting for Bayou.
Local Server is started for Bayou.
Local Server is started for Tugboat.
ERROR: Shader UI/Default shader is not
```

If all is well, you can go back to the Unity editor, make sure to switch the target platform from Server to WebGL, and then press play in the editor. Press the Client button to connect, and you should see the canvas spawn for you. You can then type whatever in the input field and press enter to send it to the server and have it write back what you wrote in the TMP text object.



"Build and run" a WebGL client to an appropriate separate folder (one we will need to zip later). This takes several minutes to build, so in its Build Settings you might want to set the "*IL2CPP Code Generation*" to "*Faster builds*" when testing. It should automatically open the game in your default browser. You can connect as a client and type text to see it's working as intended. However, this just means it's working on non-secure HTTP communication.



To test the secure WSS connection, you can host the game on a website like <https://itch.io>. You will need to zip the entire WebGL folder you built to (make sure its .zip and not .rar if using WinRAR). Start a new project and in the "*Edit Game*" page on itch.io, make sure you set the "*Kind of project*" to "*HTML*", upload the zip, and when its finished uploading, just under it check the "*This file will be played in the browser*" box.

Kind of project

HTML — You have a ZIP or HTML file that will be played in the browser ▼

TIP You can add additional downloadable files for any of the types above

Release status

Prototype — An early prototype for testing an idea out, fate of project unknown ▼

Pricing

☐ \$0 or donate

☐ Paid

☒ No payments

The project's files will be freely available and no donations can be made

Uploads

Upload a ZIP file containing your game. There must be an `index.html` file in the ZIP. Or upload a `.html` file that contains your entire game. [Learn more](#) →

Any additional files you upload will be made available for download. You can apply a minimum price to the project after uploading additional downloadable files.

WebGL.zip

[More...](#)

[Delete file](#)


8mb • [Change display name](#)

Yesterday at 5:30 PM

☒ This file will be played in the browser

TIP Use **butler** to upload game files: it only uploads what's changed, generates patches for the [itch.io app](#), and you can automate it. [Get started!](#)

Upload files

or  Choose from Dropbox

[Add External file](#) ?

File size limit: 1 GB. [Contact us](#) if you need more space

Embed options

How should your project be run in your page?

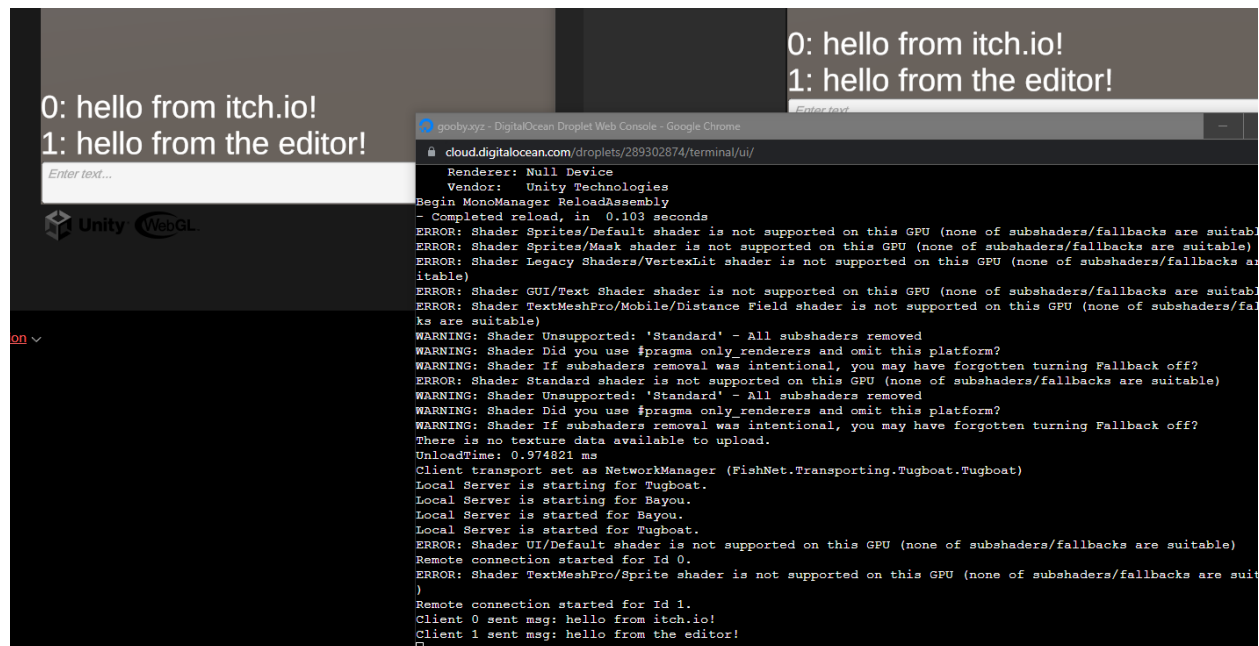
Embed in page ▼

Manually set size ▼

Viewport dimensions

Width px × Height px

Save and view the page, run your game, connect, and type! You can also connect via the editor, PC build, android build, send a build to a friend, etc. and communicate between clients.



Conclusion

There we have it, now you can go on and make that browser based MMO you've always wanted to. If you are having any issues, I urge you to please re-read through this guide. I know it's kind of long but the answer is probably in here in one way or another. If worse comes to worse ask me, gooby, and others in [FishNet's Discord](#) help channels.