

CPSC 491

CSU – Fullerton

# VRecover

Daniel Currey

Jonathan Mai

Dan Solis

Jason Chiu

Kaixiang Liu

# Goal

- Enhance the effectiveness and accessibility of exposure therapy for patients and therapists.
- Provide a cost-effective platform with home-based therapy options.
- Highly customizable scenarios for patients.
- Prioritize security and privacy of patient data, following principles from Zala et al.'s study (2022) on e-healthcare privacy as well as HIPAA compliance.





## Approach

- Create a VR-based therapy platform using Unity VR, AWS services, and FishNet networking to provide secure, scalable, and synchronized therapy sessions.

# Technology

Type	Software
Programming Language	C# (for backend development), TypeScript (for frontend development)
IDE	Visual Studio (for backend with C# .net core), Unity (for VR environment development)
Operating System	Windows 10/11 (32 or 64 bit) (compatible with Visual Studio and Unity)
Web Technologies	TypeScript, HTML5, CSS3 (with Tailwind CSS for styling)
Software Framework	Unity (for VR content creation and management), AWS (for cloud service)

# Bootstrap

```
using System;
using UnityEngine;

namespace VRecover
{
    // references
    public interface IInitializable
    {
        // references
        void Initialize();
    }

    // Unity Script 10 references
    public class Bootstrap : MonoBehaviour
    {
        public static bool isInitialized = false;
        private static GameObject servicesContainer;

        [RuntimeInitializeOnLoadMethod(RuntimeInitializeLoadType.BeforeSceneLoad)]
        // references
        static void OnBeforeSceneLoad()
        {
            Debug.Log("[Bootstrap] Starting initialization...");
            if (isInitialized)
            {
                Debug.LogWarning("[Bootstrap] Bootstrap is already initialized. Skipping additional initialization.");
                return;
            }

            Debug.Log("[Bootstrap] Cleaning up existing services...");
            CleanupExistingServices();

            Debug.Log("[Bootstrap] Registering services from prefabs...");
            if (!RegisterServicesFromPrefabs())
            {
                Debug.LogWarning("[Bootstrap] Some services could not be registered properly. Check the logs for details.");
            }

            isInitialized = true;
            Debug.Log("[Bootstrap] Bootstrap initialization completed successfully.");
        }

        // references
        private static void CleanupExistingServices()
        {
            Debug.Log("[Bootstrap] Starting cleanup of existing services...");
            // Clean up any existing services container
            if (servicesContainer != null)
            {
                Debug.Log("[Bootstrap] Destroying existing services container...");
                GameObject.DestroyImmediate(servicesContainer);
            }

            var existingContainer = GameObject.Find("Services Container");
            if (existingContainer != null)
            {
                Debug.Log("[Bootstrap] Found and destroying legacy services container...");
                GameObject.DestroyImmediate(existingContainer);
            }

            // Reset ServiceLocator
            Debug.Log("[Bootstrap] Resetting ServiceLocator...");
            ServiceLocator.Reset();
            isInitialized = false;
            Debug.Log("[Bootstrap] Cleanup completed successfully.");
        }

        // references
        private static bool RegisterServicesFromPrefabs()
        {
            // references
            Debug.Log("[Bootstrap] Starting service registration from prefabs...");
            bool allServicesRegistered = true;
            try
            {
                if (!System.IO.Directory.Exists("Assets/Resources/Services"))
                {
                    Debug.LogError("[Bootstrap] Services directory not found at Assets/Resources/Services");
                    return false;
                }

                Debug.Log("[Bootstrap] Loading service prefabs from Resources...");
                GameObject[] servicePrefabs = Resources.LoadAll<GameObject>("Services");
                if (servicePrefabs == null || servicePrefabs.Length == 0)
                {
                    Debug.LogError("[Bootstrap] No service prefabs found in Resources/Services. Ensure prefabs are properly placed.");
                    return false;
                }

                Debug.Log($"[Bootstrap] Found {servicePrefabs.Length} service prefabs to process.");
                // Sort prefabs alphabetically by name, including the letter prefix
                Array.Sort(servicePrefabs, (a, b) => string.Compare(a.name, b.name, StringComparison.OrdinalIgnoreCase));
                servicesContainer = new GameObject("Services Container");
                DontDestroyOnLoad(servicesContainer);
                Debug.Log("[Bootstrap] Created new Services Container.");

                foreach (GameObject prefab in servicePrefabs)
                {
                    if (prefab == null)
                    {
                        Debug.LogError("[Bootstrap] Null prefab found in Services folder");
                        continue;
                    }

                    try
                    {
                        Debug.Log($"[Bootstrap] Processing service prefab: {prefab.name}");
                        GameObject serviceObject = Instantiate(prefab, servicesContainer.transform);
                        MonoBehaviour[] services = serviceObject.GetComponents<MonoBehaviour>();
                        if (services == null || services.Length == 0)
                        {
                            Debug.LogError("[Bootstrap] Service prefab {prefab.name} does not have any MonoBehaviour components");
                            allServicesRegistered = false;
                            continue;
                        }

                        bool serviceRegistered = false;
                        foreach (MonoBehaviour service in services)
                        {
                            if (service is IInitializable initializable)
                            {
                                try
                                {
                                    Debug.Log($"[Bootstrap] Initializing service: {service.GetType().Name}");
                                    initializable.Initialize();
                                }
                                catch (Exception ex)
                                {
                                    // references
                                }
                            }
                        }
                    }
                }
            }
            catch (Exception ex)
            {
                // references
            }
            return allServicesRegistered;
        }
    }
}
```

```

// references
private static bool RegisterServicesFromPrefabs()
{
    Debug.Log("[Bootstrap] Starting service registration from prefabs...");
    bool allServicesRegistered = true;
    try
    {
        if (!System.IO.Directory.Exists("Assets/Resources/Services"))
        {
            Debug.LogError("[Bootstrap] Services directory not found at Assets/Resources/Services");
            return false;
        }

        Debug.Log("[Bootstrap] Loading service prefabs from Resources...");
        GameObject[] servicePrefabs = Resources.LoadAll<GameObject>("Services");
        if (servicePrefabs == null || servicePrefabs.Length == 0)
        {
            Debug.LogError("[Bootstrap] No service prefabs found in Resources/Services. Ensure prefabs are properly placed.");
            return false;
        }

        Debug.Log($"[Bootstrap] Found {servicePrefabs.Length} service prefabs to process.");
        // Sort prefabs alphabetically by name, including the letter prefix
        Array.Sort(servicePrefabs, (a, b) => string.Compare(a.name, b.name, StringComparison.OrdinalIgnoreCase));
        servicesContainer = new GameObject("Services Container");
        DontDestroyOnLoad(servicesContainer);
        Debug.Log("[Bootstrap] Created new Services Container.");

        foreach (GameObject prefab in servicePrefabs)
        {
            if (prefab == null)
            {
                Debug.LogError("[Bootstrap] Null prefab found in Services folder");
                continue;
            }

            try
            {
                Debug.Log($"[Bootstrap] Processing service prefab: {prefab.name}");
                GameObject serviceObject = Instantiate(prefab, servicesContainer.transform);
                MonoBehaviour[] services = serviceObject.GetComponents<MonoBehaviour>();
                if (services == null || services.Length == 0)
                {
                    Debug.LogError("[Bootstrap] Service prefab {prefab.name} does not have any MonoBehaviour components");
                    allServicesRegistered = false;
                    continue;
                }

                bool serviceRegistered = false;
                foreach (MonoBehaviour service in services)
                {
                    if (service is IInitializable initializable)
                    {
                        try
                        {
                            Debug.Log($"[Bootstrap] Initializing service: {service.GetType().Name}");
                            initializable.Initialize();
                        }
                        catch (Exception ex)
                        {
                            // references
                        }
                    }
                }
            }
            catch (Exception ex)
            {
                // references
            }
        }
    }
    return allServicesRegistered;
}

```

# Service Locator

```
using System;
using System.Collections.Generic;
using UnityEngine;

namespace VRRecover
{
    24 references
    public class ServiceLocator
    {
        19 references
        private static ServiceLocator _instance;
        public static ServiceLocator Instance => _instance ??= new ServiceLocator();

        private readonly Dictionary<Type, object> _services = new();

        1 reference
        private ServiceLocator() { } // Private constructor for singleton

        1 reference
        public static void Reset()
        {
            _instance = null;
        }

        0 references
        public void RegisterService<TInterface, TImplementation>(TImplementation implementation)
            where TImplementation : class, TInterface
        {
            _services[typeof(TInterface)] = implementation;
        }

        18 references
        public TInterface GetService<TInterface>() where TInterface : class
        {
            if (_services.TryGetValue(typeof(TInterface), out object service))
            {
                return (TInterface)service;
            }
            Debug.LogError($"Service of type {typeof(TInterface)} is not registered.");
            return null;
        }

        1 reference
        public void RegisterService(Type interfaceType, object implementation)
        {
            if (!interfaceType.IsInstanceOfType(implementation))
            {
                throw new ArgumentException($"Implementation must implement {interfaceType.Name}", nameof(implementation));
            }
            _services[interfaceType] = implementation;
        }
    }
}
```

# Game Manager Service

```
using System;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;
using FishNet.Managing;

namespace VRecover
{
    /// 
    /// We reference
    /// 
    public interface IGameManagerService
    {
        #references
        ServerStateType CurrentServerState { get; }
        #references
        ClientStateType CurrentClientState { get; }
        #references
        SessionStateType CurrentSessionState { get; }
        #references
        SceneStateType CurrentSceneState { get; }

        event Action<ServerStateType> OnServerStateChanged;
        event Action<ClientStateType> OnClientStateChanged;
        event Action<SessionStateType> OnSessionStateChanged;
        event Action<SceneStateType> OnSceneStateChanged;

        #references
        void UpdateServerState(ServerStateType newState);
        #references
        void UpdateClientState(ClientStateType newState);
        #references
        void UpdateSessionState(SessionStateType newState);
        #references
        void UpdateSceneState(SceneStateType newState);

        #references
        void ReloadScene();
        #references
        void SwitchToScene(string sceneName, bool updateCurrentSceneNum);
        #references
        void SwitchToScene(int sceneNum, bool updateCurrentSceneNum);
        #references
        void EndGame();

        #references
        NetworkManager NetworkManager { get; }
        #references
        NetworkStateManager NetworkStateManager { get; }

        [Service]
        /// 
        /// We have 1 or more references | references
        /// 
        public class GameManagerService : MonoBehaviour, IGameManagerService, IInitializable
        {
            #references
            public ServerStateType CurrentServerState { get; private set; }
            #references
            public ClientStateType CurrentClientState { get; private set; }
            #references
            public SessionStateType CurrentSessionState { get; private set; }
            #references
            public SceneStateType CurrentSceneState { get; private set; }

            public event Action<ServerStateType> OnServerStateChanged;
            public event Action<ClientStateType> OnClientStateChanged;
            public event Action<SessionStateType> OnSessionStateChanged;
            public event Action<SceneStateType> OnSceneStateChanged;

            private IGameState _currentServerState;
            private IGameState _currentClientState;
            private IGameState _currentSessionState;
            private IGameState _currentSceneState;

            private readonly Dictionary<ServerStateType, IGameState> _serverStates = new();
            private readonly Dictionary<ClientStateType, IGameState> _clientStates = new();
            private readonly Dictionary<SessionStateType, IGameState> _sessionStates = new();
            private readonly Dictionary<SceneStateType, IGameState> _sceneStates = new();

            private readonly List<string> _stateTransitionHistory = new();

            private NetworkStateSync _networkStateSync;
            private NetworkManager _networkManager;
            private NetworkStateManager _networkStateManager;

            #references
            public NetworkManager NetworkManager => _networkManager;
            #references
            public NetworkStateManager NetworkStateManager => _networkStateManager;

            #references
            public void Initialize()
            {
                #references
                public void Start()
                {
                    _networkManager = GameObject.FindFirstObjectByType<NetworkManager>();
                    if (_networkManager == null)
                    {
                        Debug.LogError("GameManagerService: NetworkManager not found in scene!");
                        return;
                    }

                    _networkStateManager = new NetworkStateManager(_networkManager);

                    QualitySettings.vSyncCount = 0;

                    _networkStateSync = FindFirstObjectByType<NetworkStateSync>();
                    if (_networkStateSync == null)
                    {
                        Debug.Log("GameManagerService: NetworkStateSync not found, creating new instance");
                        var networkObject = new GameObject("NetworkStateSync");
                        _networkStateSync = networkObject.AddComponent<NetworkStateSync>();
                    }

                    InitializeStateDictionaries();
                    SetInitialStates();
                    Debug.Log("GameManagerService: Initialization complete");
                }

                #references
                private void InitializeStateDictionaries()
                {
                    try
                    {
                        Debug.Log("GameManagerService: Initializing state dictionaries");

                        // Initialize server states
                        foreach (ServerStateType stateType in Enum.GetValues<ServerStateType>())
                        {
                            _serverStates[stateType] = GameStateFactory.CreateState(stateType);
                        }

                        // Initialize client states
                        foreach (ClientStateType stateType in Enum.GetValues<ClientStateType>())
                        {
                            _clientStates[stateType] = GameStateFactory.CreateState(stateType);
                        }

                        // Initialize session states
                        foreach (SessionStateType stateType in Enum.GetValues<SessionStateType>())
                        {
                            _sessionStates[stateType] = GameStateFactory.CreateState(stateType);
                        }

                        // Initialize scene states
                        foreach (SceneStateType stateType in Enum.GetValues<SceneStateType>())
                        {
                            _sceneStates[stateType] = GameStateFactory.CreateState(stateType);
                        }

                        Debug.Log("GameManagerService: All state dictionaries initialized successfully");
                    }
                    catch (Exception ex)
                    {
                        Debug.LogError($"Failed to initialize state dictionaries: {ex.Message}");
                        Debug.LogError($"Stack trace: {ex.StackTrace}");
                        throw;
                    }
                }

                #references
                private void SetInitialStates()
                {
                    Debug.Log("GameManagerService: Setting initial states");

                    // Set the num values without triggering state changes first
                    CurrentServerState = ServerStateType.DEFAULT;
                    CurrentClientState = ClientStateType.DEFAULT;
                    CurrentSessionState = SessionStateType.DEFAULT;
                    CurrentSceneState = SceneStateType.DEFAULT;

                    // Now manually trigger the initial state entries
                    _currentServerState = _serverStates[ServerStateType.DEFAULT];
                    _currentClientState = _clientStates[ClientStateType.DEFAULT];
                    _currentSessionState = _sessionStates[SessionStateType.DEFAULT];
                    _currentSceneState = _sceneStates[SceneStateType.DEFAULT];

                    // Call OnEnter for each initial state
                    Debug.Log("GameManagerService: Triggering initial state entries");
                    _currentServerState.OnEnter(this);
                    _currentClientState.OnEnter(this);
                    _currentSessionState.OnEnter(this);
                    _currentSceneState.OnEnter(this);

                    // Notify listeners of initial states
                    OnServerStateChanged?.Invoke(CurrentServerState);
                    OnClientStateChanged?.Invoke(CurrentClientState);
                    OnSessionStateChanged?.Invoke(CurrentSessionState);
                    OnSceneStateChanged?.Invoke(CurrentSceneState);

                    Debug.Log("GameManagerService: Initial states set and entered");
                }
            }
        }
    }
}
```

```
private readonly Dictionary<ServerStateType, IGameState> _serverStates = new();
private readonly Dictionary<ClientStateType, IGameState> _clientStates = new();
private readonly Dictionary<SessionStateType, IGameState> _sessionStates = new();
private readonly Dictionary<SceneStateType, IGameState> _sceneStates = new();

private readonly List<string> _stateTransitionHistory = new();

private NetworkStateSync _networkStateSync;
private NetworkManager _networkManager;
private NetworkStateManager _networkStateManager;

#references
public NetworkManager NetworkManager => _networkManager;
#references
public NetworkStateManager NetworkStateManager => _networkStateManager;

#references
public void Initialize()
{
    #references
    public void Start()
    {
        _networkManager = GameObject.FindFirstObjectByType<NetworkManager>();
        if (_networkManager == null)
        {
            Debug.LogError("GameManagerService: NetworkManager not found in scene!");
            return;
        }

        _networkStateManager = new NetworkStateManager(_networkManager);

        QualitySettings.vSyncCount = 0;

        _networkStateSync = FindFirstObjectByType<NetworkStateSync>();
        if (_networkStateSync == null)
        {
            Debug.Log("GameManagerService: NetworkStateSync not found, creating new instance");
            var networkObject = new GameObject("NetworkStateSync");
            _networkStateSync = networkObject.AddComponent<NetworkStateSync>();
        }

        InitializeStateDictionaries();
        SetInitialStates();
        Debug.Log("GameManagerService: Initialization complete");
    }

    #references
    private void InitializeStateDictionaries()
    {
        try
        {
            Debug.Log("GameManagerService: Initializing state dictionaries");

            // Initialize server states
            foreach (ServerStateType stateType in Enum.GetValues<ServerStateType>())
            {
                _serverStates[stateType] = GameStateFactory.CreateState(stateType);
            }

            // Initialize client states
            foreach (ClientStateType stateType in Enum.GetValues<ClientStateType>())
            {
                _clientStates[stateType] = GameStateFactory.CreateState(stateType);
            }

            // Initialize session states
            foreach (SessionStateType stateType in Enum.GetValues<SessionStateType>())
            {
                _sessionStates[stateType] = GameStateFactory.CreateState(stateType);
            }

            // Initialize scene states
            foreach (SceneStateType stateType in Enum.GetValues<SceneStateType>())
            {
                _sceneStates[stateType] = GameStateFactory.CreateState(stateType);
            }

            Debug.Log("GameManagerService: All state dictionaries initialized successfully");
        }
        catch (Exception ex)
        {
            Debug.LogError($"Failed to initialize state dictionaries: {ex.Message}");
            Debug.LogError($"Stack trace: {ex.StackTrace}");
            throw;
        }
    }

    #references
    private void SetInitialStates()
    {
        Debug.Log("GameManagerService: Setting initial states");

        // Set the num values without triggering state changes first
        CurrentServerState = ServerStateType.DEFAULT;
        CurrentClientState = ClientStateType.DEFAULT;
        CurrentSessionState = SessionStateType.DEFAULT;
        CurrentSceneState = SceneStateType.DEFAULT;

        // Now manually trigger the initial state entries
        _currentServerState = _serverStates[ServerStateType.DEFAULT];
        _currentClientState = _clientStates[ClientStateType.DEFAULT];
        _currentSessionState = _sessionStates[SessionStateType.DEFAULT];
        _currentSceneState = _sceneStates[SceneStateType.DEFAULT];

        // Call OnEnter for each initial state
        Debug.Log("GameManagerService: Triggering initial state entries");
        _currentServerState.OnEnter(this);
        _currentClientState.OnEnter(this);
        _currentSessionState.OnEnter(this);
        _currentSceneState.OnEnter(this);

        // Notify listeners of initial states
        OnServerStateChanged?.Invoke(CurrentServerState);
        OnClientStateChanged?.Invoke(CurrentClientState);
        OnSessionStateChanged?.Invoke(CurrentSessionState);
        OnSceneStateChanged?.Invoke(CurrentSceneState);

        Debug.Log("GameManagerService: Initial states set and entered");
    }
}
```

```
try
{
    Debug.Log("GameManagerService: Initializing state dictionaries");

    // Initialize server states
    foreach (ServerStateType stateType in Enum.GetValues<ServerStateType>())
    {
        _serverStates[stateType] = GameStateFactory.CreateState(stateType);
    }

    // Initialize client states
    foreach (ClientStateType stateType in Enum.GetValues<ClientStateType>())
    {
        _clientStates[stateType] = GameStateFactory.CreateState(stateType);
    }

    // Initialize session states
    foreach (SessionStateType stateType in Enum.GetValues<SessionStateType>())
    {
        _sessionStates[stateType] = GameStateFactory.CreateState(stateType);
    }

    // Initialize scene states
    foreach (SceneStateType stateType in Enum.GetValues<SceneStateType>())
    {
        _sceneStates[stateType] = GameStateFactory.CreateState(stateType);
    }

    Debug.Log("GameManagerService: All state dictionaries initialized successfully");
}
catch (Exception ex)
{
    Debug.LogError($"Failed to initialize state dictionaries: {ex.Message}");
    Debug.LogError($"Stack trace: {ex.StackTrace}");
    throw;
}

#references
private void SetInitialStates()
{
    Debug.Log("GameManagerService: Setting initial states");

    // Set the num values without triggering state changes first
    CurrentServerState = ServerStateType.DEFAULT;
    CurrentClientState = ClientStateType.DEFAULT;
    CurrentSessionState = SessionStateType.DEFAULT;
    CurrentSceneState = SceneStateType.DEFAULT;

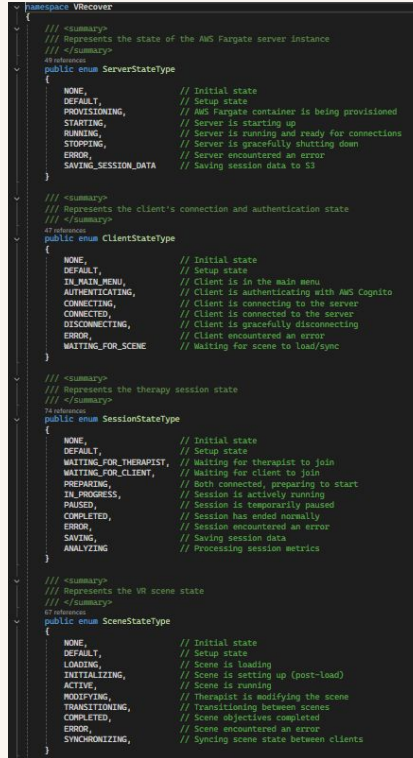
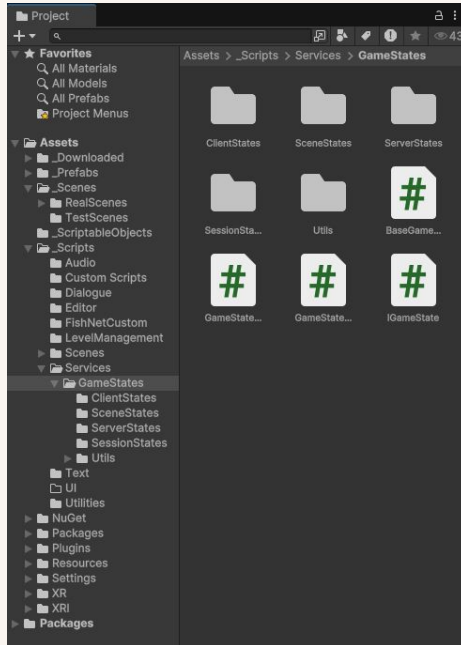
    // Now manually trigger the initial state entries
    _currentServerState = _serverStates[ServerStateType.DEFAULT];
    _currentClientState = _clientStates[ClientStateType.DEFAULT];
    _currentSessionState = _sessionStates[SessionStateType.DEFAULT];
    _currentSceneState = _sceneStates[SceneStateType.DEFAULT];

    // Call OnEnter for each initial state
    Debug.Log("GameManagerService: Triggering initial state entries");
    _currentServerState.OnEnter(this);
    _currentClientState.OnEnter(this);
    _currentSessionState.OnEnter(this);
    _currentSceneState.OnEnter(this);

    // Notify listeners of initial states
    OnServerStateChanged?.Invoke(CurrentServerState);
    OnClientStateChanged?.Invoke(CurrentClientState);
    OnSessionStateChanged?.Invoke(CurrentSessionState);
    OnSceneStateChanged?.Invoke(CurrentSceneState);

    Debug.Log("GameManagerService: Initial states set and entered");
}
```

# Game States





# AWS Login Manager

```
C:\Users\Dan\Documents\Git Desktop Projects\wrecover\Assets\Custom Scripts\LoginManagers
1 using System.Collections;
2 using System.Threading;
3 using System.Threading.Tasks;
4 using UnityEngine;
5 using UnityEngine.UI;
6 using Amazon;
7 using Amazon.CognitoIdentityProvider;
8 using Amazon.Extensions.CognitoAuthentication;
9 using TMPPro;
10 using UnityEngine.SceneManagement;
11
12
13 public class LoginManager : MonoBehaviour
14 {
15     public TMP_InputField usernameInput;
16     public TMP_InputField passwordInput;
17     public Button loginButton;
18     public TextMeshProUGUI errorText;
19     //public GameObject mainUI; // The main UI to enable after login --// no longer used
20
21     private AmazonCognitoIdentityProviderClient _provider;
22     private CognitoUserPool _userPool;
23
24     private string userPoolId = " "; // Replace with your User Pool ID
25     private string clientId = " "; // Replace with your App Client ID
26     private string region = " "; // Change to your region
27
28     void Start()
29     {
30         // Initialize the provider and user pool
31         _provider = new AmazonCognitoIdentityProviderClient(new Amazon.Runtime.AnonymousAWSCredentials(), RegionEndpoint.GetBySystemName(region));
32         _userPool = new CognitoUserPool(userPoolId, clientId, _provider);
33
34         // Disable main UI at start
35         // mainUI.SetActive(false);
36
37         // Set up login button listener
38         loginButton.onClick.AddListener(() => StartCoroutine(AttemptLogin()));
39     }
40
41     private IEnumerator AttemptLogin()
42     {
43         // Retrieve username and password from the input fields
44         string username = usernameInput.text;
45         string password = passwordInput.text;
46
47         // Check if the fields are empty
48         if (string.IsNullOrEmpty(username) || string.IsNullOrEmpty(password))
49         {
```

# AWS User Custom Attribute

**User: dev** [Info](#) [Actions](#)

**User information**

User ID (Sub)  
91fbc540-6081-7049-c4da-efc661f8953

Account status  
Enabled

Confirmation status  
Confirmed

Created time  
November 18, 2024 at 16:40 PST

Last updated time  
November 25, 2024 at 02:01 PST

MFA setting  
MFA inactive

MFA methods  
-

**User attributes (2)** [Info](#) [Edit](#)

View and edit this user's attributes.

Attribute name	Value	Type
custom:role	therapist	Optional
sub	91fbc540-6081-7049-c4da-efc661f8953	Required

**Group memberships (0)** [Info](#) [Remove user from group](#) [Add user to group](#)

View and edit this user's group memberships.

Group name	Description	Group created time
------------	-------------	--------------------

**User: client** [Info](#)

**User information**

User ID (Sub)  
f12bd510-f081-70fc-e5f0-46628277e78f

Account status  
Enabled

Confirmation status  
Confirmed

Created time  
November 25, 2024 at 10:13 PST

Last updated time  
December 9, 2024 at 02:32 PST

MFA setting  
MFA inactive

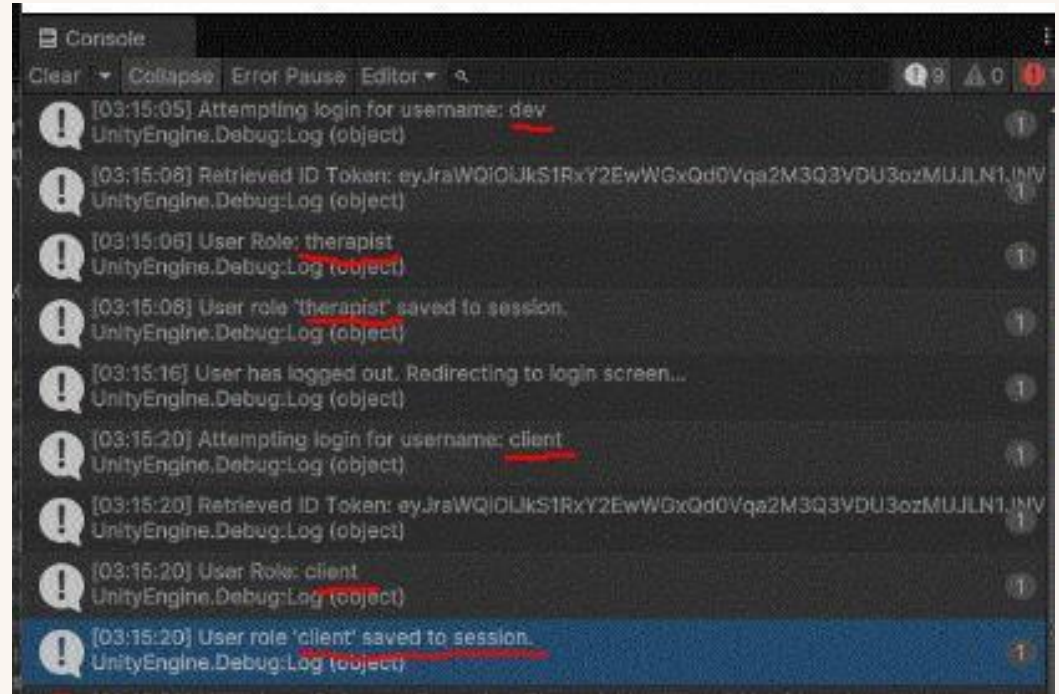
MFA methods  
-

**User attributes (2)** [Info](#)

View and edit this user's attributes.

Attribute name	Value	Type
custom:role	client	Optional
sub	f12bd510-f081-70fc-e5f0-46628277e78f	Required

# Attribute Reading



The screenshot shows a console window with the following logs:

```
[03:15:05] Attempting login for username: dev
UnityEngine.Debug:Log (object)

[03:15:08] Retrieved ID Token: eyJraWQlOiJkS1RyY2EwWGxQd0Vqa2M3Q3VDU3ozMUJLN1INV
UnityEngine.Debug:Log (object)

[03:15:08] User Role: therapist
UnityEngine.Debug:Log (object)

[03:15:08] User role 'therapist' saved to session.
UnityEngine.Debug:Log (object)

[03:15:16] User has logged out. Redirecting to login screen...
UnityEngine.Debug:Log (object)

[03:15:20] Attempting login for username: client
UnityEngine.Debug:Log (object)

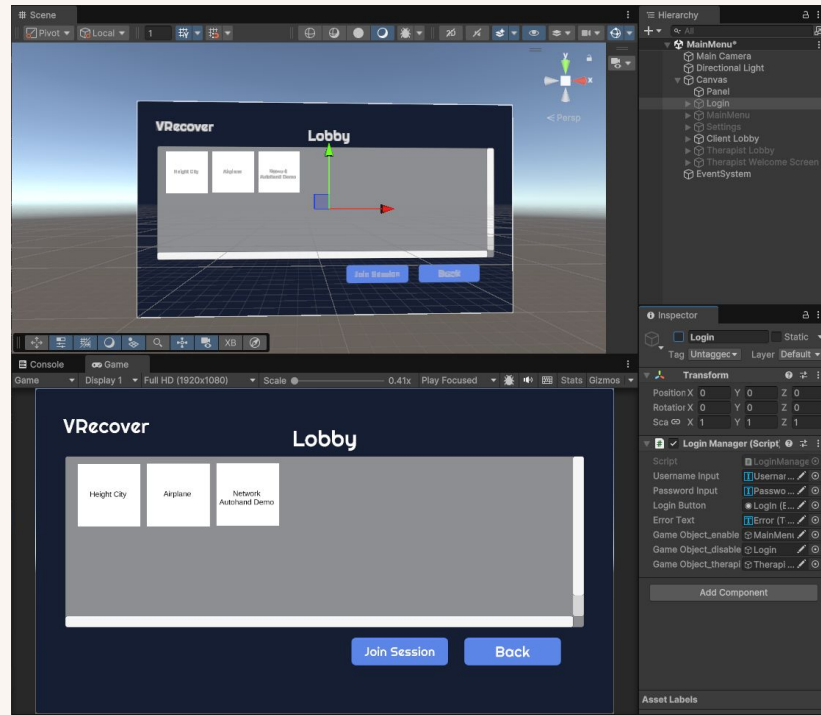
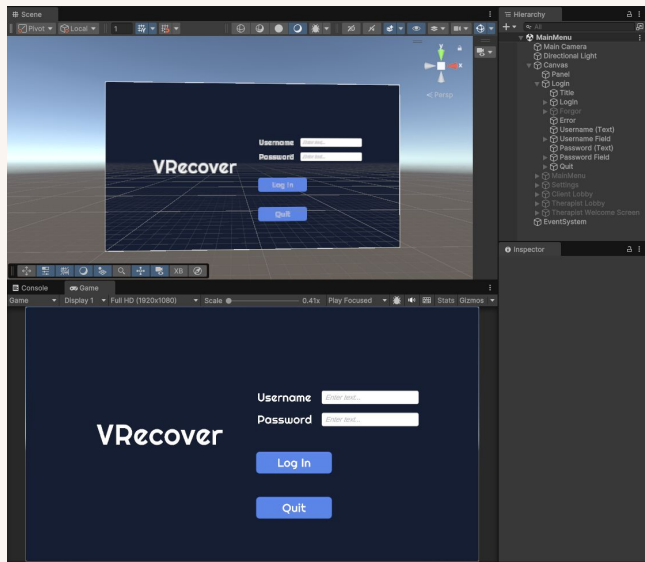
[03:15:20] Retrieved ID Token: eyJraWQlOiJkS1RyY2EwWGxQd0Vqa2M3Q3VDU3ozMUJLN1INV
UnityEngine.Debug:Log (object)

[03:15:20] User Role: client
UnityEngine.Debug:Log (object)

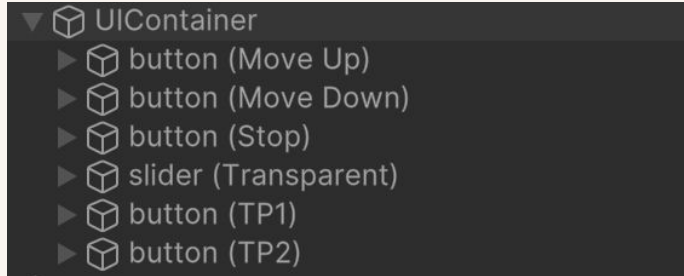
[03:15:20] User role 'client' saved to session.
UnityEngine.Debug:Log (object)
```

The console window has a title bar "Console" and buttons for "Clear", "Collapse", "Error Pause", and "Editor". There are also icons for search, refresh, and a red error icon. The logs are displayed in a dark theme with yellow exclamation mark icons on the left and a column of numbers on the right.

# Menu



# Serialized UI



```
1 {  
2   "scene": {  
3     "name": "jon-test-scene",  
4     "elements": [  
5       {  
6         "type": "button",  
7         "label": "Move Up",  
8         "targetObject": "Elevator",  
9         "method": "MoveElevatorUp"  
10      },  
11      {  
12        "type": "button",  
13        "label": "Move Down",  
14        "targetObject": "Elevator",  
15        "method": "MoveElevatorDown"  
16      },  
17      {  
18        "type": "button",  
19        "label": "Stop",  
20        "targetObject": "Elevator",  
21        "method": "StopElevator"  
22      },  
23      {  
24        "type": "slider",  
25        "label": "Transparent",  
26        "min": 0,  
27        "max": 1,  
28        "targetObject": "Elevator",  
29        "property": "Renderer.material.color.a"  
30      },  
31      {  
32        "type": "button",  
33        "label": "TP1",  
34        "targetObject": "TransportManager",  
35        "method": "TeleportToSpawnPoint",  
36        "parameters": [ 0 ]  
37      },  
38      {  
39        "type": "button",  
40        "label": "TP2",  
41        "targetObject": "TransportManager",  
42        "method": "TeleportToSpawnPoint",  
43        "parameters": [ 1 ]  
44      }  
45    ]  
46  }  
47 }  
48 }
```

# DynamicUIManager

```

1  using UnityEngine;
2  using UnityEditor;
3  using System;
4  using UnityEngine.UI;
5  using System.Collections;
6
7  public class DynamicUIManager : MonoBehaviour
8  {
9      [Header("Prefabs")]
10     [SerializeField] private GameObject sliderPrefab;
11     [SerializeField] private GameObject togglePrefab;
12     [SerializeField] private GameObject buttonPrefab;
13
14     [Header("UI Container (Design Canvas Here)")]
15     [SerializeField] private Transform canvas;
16
17     [Header("Metadata Files")]
18     [SerializeField] private string metadataFilename = "metadata";
19
20     private Transform uiContainer;
21
22     void Start()
23     {
24         SetupUIContainer();
25
26         LoadSceneMetadata(sceneMetadata = LoadMetadata(metadataFilename));
27         if (sceneMetadata == null)
28         {
29             Debug.LogError("Failed to load metadata.");
30             return;
31         }
32
33         if (string.Equals(sceneMetadata.name, UnityEngine.SceneManagement.SceneManager.GetActiveScene().name, StringComparison.OrdinalIgnoreCase))
34         {
35             Debug.LogWarning("Metadata is for scene '{sceneMetadata.name}', but current scene is '{UnityEngine.SceneManagement.SceneManager.GetActiveScene().name}'.");
36             return;
37         }
38
39         foreach (var element in sceneMetadata.elements)
40         {
41             GameObject targetObj = GameObject.Find(element.targetObj);
42             if (targetObj == null)
43             {
44                 Debug.LogWarning("Target object '{element.targetObj}' not found in the scene.");
45                 continue;
46             }
47
48             CreateUIElement(element, targetObj);
49         }
50
51     private void SetupUIContainer()
52     {
53         if (uiContainer == null)
54         {
55             GameObject containerObj = new GameObject("UIContainer");
56             containerObj.transform.SetParent(canvas, false);
57             RectTransform transform = containerObj.GetComponent<RectTransform>();
58             RectTransform anchorMin = new RectTransform(0, 0);
59             RectTransform anchorMax = new RectTransform(1, 1);
60             RectTransform offsetMin = new RectTransform(0, 0);
61             RectTransform offsetMax = new RectTransform(0, 0);
62
63             var layoutGroup = containerObj.AddComponent<VerticalLayoutGroup>();
64             layoutGroup.spacing = 20;
65             layoutGroup.padding = new RectOffset(10, 10, 10, 10);
66             layoutGroup.childAlignment = TextAnchor.UpperCenter;
67
68             var contentFitter = containerObj.AddComponent<ContentFitter>();
69             contentFitter.verticalFit = ContentFitter.FitVertical;
70
71             uiContainer = containerObj.transform;
72         }
73     }
74
75     private void LoadSceneMetadata(LoadMetadata[] files)
76     {
77         foreach (var file in Resources.Load<LoadMetadata>("files"));
78         if (files != null)
79         {
80             Debug.LogError("Metadata file '{filename}' not found in Resources folder.");
81             return null;
82         }
83     }
84 }

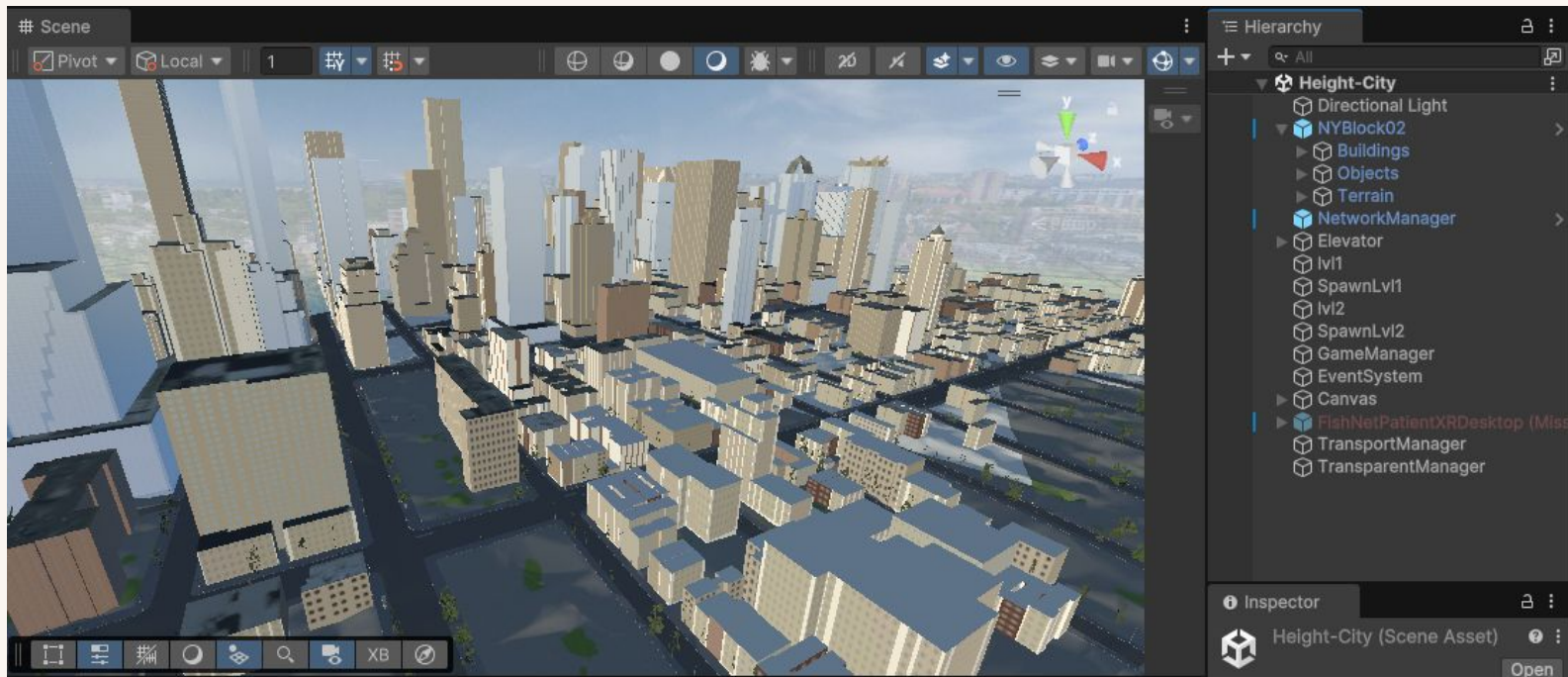
```

```

91     private void CreateUIElement(UIElement element, GameObject targetObj)
92     {
93         GameObject uiElement = null;
94
95         if (string.Equals(element.type, "slider", StringComparison.OrdinalIgnoreCase))
96         {
97             uiElement = Instantiate(sliderPrefab);
98             ConfigureSlider(uiElement, element, targetObj);
99         }
100         else if (string.Equals(element.type, "toggle", StringComparison.OrdinalIgnoreCase))
101         {
102             uiElement = Instantiate(togglePrefab);
103             ConfigureToggle(uiElement, element, targetObj);
104         }
105         else if (string.Equals(element.type, "button", StringComparison.OrdinalIgnoreCase))
106         {
107             uiElement = Instantiate(buttonPrefab);
108             ConfigureButton(uiElement, element, targetObj);
109         }
110
111         if (uiElement != null)
112         {
113             uiElement.transform.SetParent(uiContainer, false);
114             uiElement.name = $"{element.type} {element.Label ?? element.targetObj}";
115         }
116     }
117
118     private void ConfigureSlider(GameObject sliderObj, UIElement element, GameObject targetObj)
119     {
120         var label = sliderObj.transform.Find<Label>().GetComponent<Text>().text;
121         label.text = element.Label;
122
123         var slider = sliderObj.GetComponent<Slider>();
124         object currentValueObj = PropertyUtility.GetValue(targetObj, element.property);
125         if (currentValueObj is float currentValue)
126         {
127             float range = element.max - element.min;
128
129             slider.minValue = currentValue - range / 2f;
130             slider.maxValue = currentValue + range / 2f;
131             slider.value = currentValue;
132
133             slider.onValueChanged.AddListener(value =>
134             {
135                 PropertyUtility.SetValue(targetObj, element.property, value);
136             });
137         }
138         else
139         {
140             Debug.LogWarning("Property '{element.property}' is not a float and cannot be controlled by the slider.");
141         }
142     }
143
144     private void ConfigureToggle(GameObject toggleObj, UIElement element, GameObject targetObj)
145     {
146         var label = toggleObj.GetComponent<Text>().text;
147         label.text = element.Label;
148
149         var toggle = toggleObj.GetComponent<Toggle>();
150         toggle.isOn = Convert.ToBoolean(element.defaultValue);
151
152         toggle.onValueChanged.AddListener(isOn =>
153         {
154             PropertyUtility.SetValue(targetObj, element.property,.isOn);
155         });
156     }
157
158     private void ConfigureButton(GameObject buttonObj, UIElement element, GameObject targetObj)
159     {
160         var label = buttonObj.GetComponent<Text>().text;
161         label.text = element.Label;
162
163         var button = buttonObj.GetComponent<Button>();
164         button.onClick.AddListener(() =>
165         {
166             // Look for any MonoBehaviour scripts on the target object
167             var scripts = targetObj.GetComponents<MonoBehaviour>();
168             if (scripts.Length == 0)
169             {
170                 Debug.LogError("No MonoBehaviour scripts found on '{targetObj.name}'.");
171                 return;
172             }
173
174             // Loop through each script to find the specified method
175             foreach (var script in scripts)
176             {
177                 if (script.GetType().GetMethod(element.method) != null)
178                 {
179                     return;
180                 }
181             }
182         });
183     }

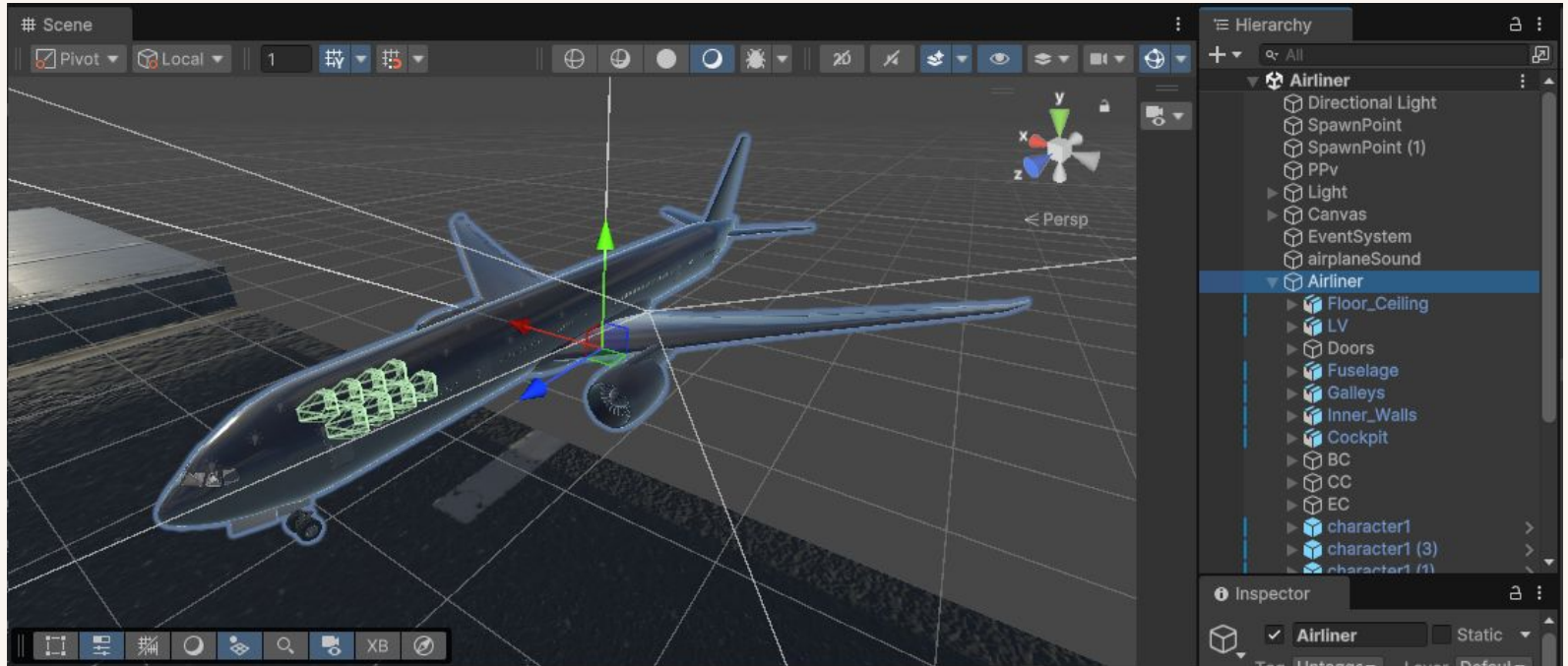
```

# Heights Scene





# Airliner Scene





# Dockerfile

```
1 # Use the available UnityCI Docker image
2 FROM unityci/editor:ubuntu-2020.2.1f1-linux-il2cpp-3.1.0
3
4 # Set the working directory inside the container
5 WORKDIR /vrecover
6
7 # Copy your project files into the container
8 COPY . .
9 | Ctrl+L to chat, Ctrl+K to generate
10 # Build the Unity project
11 ✓ RUN echo "Building Unity project..." && \
12     unity-editor -projectPath /vrecover \
13     -quit -batchmode -buildTarget linux64 \
14     -logfile /vrecover/build.log \
15     -outputPath /vrecover/Build/ServerBuild.x86_64 || \
16     echo "Unity build failed. Check /vrecover/build.log for details."
17
18 # Verify the build output
19 ✓ RUN echo "Verifying server build output..." && \
20     if [ -f "/vrecover/Build/ServerBuild.x86_64" ]; then \
21         chmod +x /vrecover/Build/ServerBuild.x86_64 && \
22         echo "Build output found and made executable."; \
23     else \
24         echo "Build output not found. Check Unity build logs."; \
25     fi
26
27 # Expose the necessary ports for FishNet
28 EXPOSE 7770/tcp
29 EXPOSE 7771/udp
30
31 # Run the server in headless mode
32 ENTRYPOINT ["/vrecover/Build/ServerBuild.x86_64", "-batchmode", "-nographics"]
```

# GitHub Actions Workflow

```
1  name: Build and Deploy Unity Server to AWS Fargate
2
3  on:
4    push:
5      branches:
6        - main
7
8  jobs:
9    deploy:
10     runs-on: ubuntu-latest
11
12     steps:
13       - name: Checkout Code
14         uses: actions/checkout@v2
15
16       - name: Configure AWS Credentials
17         uses: aws-actions/configure-aws-credentials@v1
18         with:
19           aws-access-key-id: ${ secrets.AWS_ACCESS_KEY_ID }
20           aws-secret-access-key: ${ secrets.AWS_SECRET_ACCESS_KEY }
21           aws-region: ${ secrets.AWS_REGION }
22
23       - name: Log in to Amazon ECR
24         id: login-ecr
25         uses: aws-actions/amazon-ecr-login@v1
26
27       - name: Build Docker Image
28         run: |
29           docker build -t unity-server .
30           docker tag unity-server:latest ${ secrets.ECR_REPOSITORY_URI }:latest
31
32       - name: Push Docker Image to ECR
33         run: |
34           docker push ${ secrets.ECR_REPOSITORY_URI }:latest
35
36       - name: Register ECS Task Definition
37         env:
38           ECR_IMAGE: ${ secrets.ECR_REPOSITORY_URI }:latest
39         run: |
40           # Create a new ECS task definition based on a JSON template
41           cat <<EOF > task-definition.json
42           {
43             "family": "VRRecoverUnityServer-Task",
44             "networkMode": "awsvpc",
45             "containerDefinitions": [
```

```
8    jobs:
9      deploy:
10       steps:
11         - name: Register ECS Task Definition
12           env:
13             run: |
14               # Create a new ECS task definition based on a JSON template
15               cat <<EOF > task-definition.json
16               {
17                 "family": "VRRecoverUnityServer-Task",
18                 "networkMode": "awsvpc",
19                 "containerDefinitions": [
20                   {
21                     "name": "UnityGameServer",
22                     "image": "${ ECR_IMAGE }",
23                     "essential": true,
24                     "memory": 512,
25                     "cpu": 256,
26                     "portMappings": [
27                       {
28                         "containerPort": 7770,
29                         "protocol": "tcp"
30                       },
31                       {
32                         "containerPort": 7771,
33                         "protocol": "udp"
34                       }
35                     ]
36                   }
37                 ]
38               },
39               "requiresCompatibilities": [
40                 "FARGATE"
41               ],
42               "cpu": "256",
43               "memory": "512",
44               "executionRoleArn": "arn:aws:iam::767397769064:role/ecsTaskExecutionRole"
45             EOF
46           aws ecs register-task-definition --cli-input-json file://task-definition.json
47
48         - name: Update ECS Service
49           run: |
50             aws ecs update-service --cluster ${ secrets.ECS_CLUSTER_NAME } --service ${ secrets.ECS_SERVICE_NAME } --force-new-deployment --region ${ secrets.AWS_REGION }
```

# GitHub Actions Runs

The screenshot shows the GitHub Actions interface for the repository 'spicy'. The 'All workflows' tab is selected, displaying a list of workflow runs. The left sidebar shows the repository structure with 'Build and Deploy Unity Server to AWS F...' as the active workflow. The main area shows a table of workflow runs with columns for Event, Status, Branch, and Actor. The runs are filtered by 'Filter workflow runs'.

Event	Status	Branch	Actor
Merge pull request #28 from spicy/dev	Failed	main	pm307
Resolved all .meta file conflicts from main	Success	main	pm307
Merge pull request #17 from spicy/dev	Success	main	pm307
Resolved all .meta file conflicts from main	Success	main	pm307
Merge pull request #17 from spicy/dev	Success	main	pm307
Merge pull request #16 from spicy/dev	Success	main	pm307
Merge pull request #15 from spicy/dev	Success	main	pm307
Merge pull request #14 from spicy/dev	Success	main	pm307
Merge pull request #13 from spicy/dev	Success	main	pm307
Merge pull request #11 from spicy/dev	Success	main	pm307
Merge pull request #10 from spicy/dev	Success	main	pm307
Merge pull request #9 from spicy/dev	Success	main	pm307
Merge pull request #6 from spicy/dev	Success	main	pm307

The screenshot shows the 'Summary' page for the workflow 'Build and Deploy Unity Server to AWS Fargate'. The 'Summary' tab is selected, displaying a list of jobs. The 'deploy' job is highlighted, showing its status as 'Success' and a duration of '1m 53s'. The 'Annotations' section shows a list of warnings. The 'Workflow file' section shows the steps of the workflow.

**Summary**

Build and Deploy Unity Server to AWS Fargate

Resolved all .meta file conflicts from main #12

Re-run all jobs

**Annotations**

3 warnings

**Jobs**

deploy

Run details

Usage

Workflow file

**Annotations**

Search logs

- Set Up job
- Checkout Code
- Configure AWS Credentials
- Log in to Amazon ECR
- Build Docker Image
- Push Docker Image to ECR
- Register ECS Task Definition
- Update ECS Service
- Post Log in to Amazon ECR
- Post Configure AWS Credentials
- Post Checkout Code
- Complete job

# AWS Repository

## Private repositories (1)

Search by repository substring

Repository name

URI



unity-vr-therapy-server



767397769064.dkr.ecr.us-west-1.amazonaws.com/unity-vr-therapy-server

Repositories > unity-vr-therapy-server > sha256:d7e09e8a5ca9dcfb0908243ea00e6aed2c89287ee1824b7bf49565a15cddc11

### Image

#### Details

**Image tags**  
latest

**URI**  
767397769064.dkr.ecr.us-west-1.amazonaws.com/unity-vr-therapy-server:latest

**Digest**  
sha256:d7e09e8a5ca9dcfb0908243ea00e6aed2c89287ee1824b7bf49565a15cddc11

#### General information

<b>Artifact type</b> Image	<b>Repository</b> unity-vr-therapy-server	<b>Pushed at</b> December 08, 2024, 16:10:37 (UTC-07)
<b>Size (MiB)</b> 639.96		

#### Scanning and vulnerabilities

Status  
Scan not found

Scan

#### Referrers [Info](#)

Referrer digest	Type	Pushed at
No referrers This artifact does not have any referrers		

#### Replication status [Info](#)

Select status source  
Queries for replication status based on selection of a specific tag or by the image digest

digest

Filter status

Target account	Region	Status	Error
No replication status Queried tag or digest does not have replication statuses			

# AWS Fargate, Clusters

The screenshot displays the AWS Elastic Container Service (ECS) console interface. The top navigation bar shows the path: Amazon Elastic Container Service > Clusters > VRecoverUnityServerCluster > Services. The cluster name "VRecoverUnityServerCluster" is prominently displayed at the top of the main content area, along with its last updated timestamp (December 09, 2024 at 16:25 UTC-7:00) and buttons for "Update cluster" and "Delete cluster".

The "Cluster overview" section provides a summary of the cluster's state:

- ARN:** arn:aws:ecs:us-west-1:767397769064:cluster/VRecoverUnityServerCluster
- Status:** Active (indicated by a green checkmark)
- CloudWatch monitoring:** Default (checked)
- Registered container instances:** -
- Services:**
  - Draining: -
  - Active: 1
- Tasks:**
  - Pending: -
  - Running: -
- Encryption:**
  - Managed storage: -
  - Fargate ephemeral storage: -

Below the overview, there are tabs for "Services", "Tasks", "Infrastructure", "Metrics", "Scheduled tasks", and "Tags". The "Services" tab is currently selected, showing a list of services under the heading "Services (1) Info".

The services list includes a search bar "Filter services by value" and filters for "Filter launch type" (Any launch type) and "Filter service type" (Any service type). The table below lists the services:

<input type="checkbox"/>	Service name	ARN	Status	Service type	Deployments and tasks	Last deployment	Task definition	Launch type
<input type="checkbox"/>	VRecoverUnityServer-Service	arn:aws.ecs:us-west-1:767397769064:cluster/VRecoverUnityServer-cluster/VRecoverUnityServer-Service	Active	REPLICA	0/1 Tasks running	Failed	VRecoverUnityServe...	-

# AWS Fargate Task Definitions

Amazon Elastic Container Service

Task definitions (1) [Info](#)

Filter by status: Active

Search: Filter task definitions

Task definition: [VRecoverUnityServer-Task](#) ACTIVE

Status of last revision: ACTIVE

Buttons: Deploy, Create new revision, Create new task definition

Account settings: Updated

VRecoverUnityServer-Task:5

Buttons: Deploy, Actions, Create new revision

Overview [Info](#)

ARN: [arn:aws:ecs:us-west-1:767397769064:task-definition/VRecoverUnityServer-Task:5](#)

Status: ACTIVE

Time created: December 08, 2024 at 16:10 (UTC-7:00)

App environment: Fargate

Task role: [ecsTaskExecutionRole](#)

Operating system/Architecture: -

Network mode: awsvpc

Containers: JSON Task placement Volumes (0) Requires attributes Tags

Task size

Task CPU: 256 units (0.25 vCPU)

Task CPU maximum allocation for containers

CPU (unit) bar chart: 0 to 240

Task memory: 512 MiB (0.5 GiB)

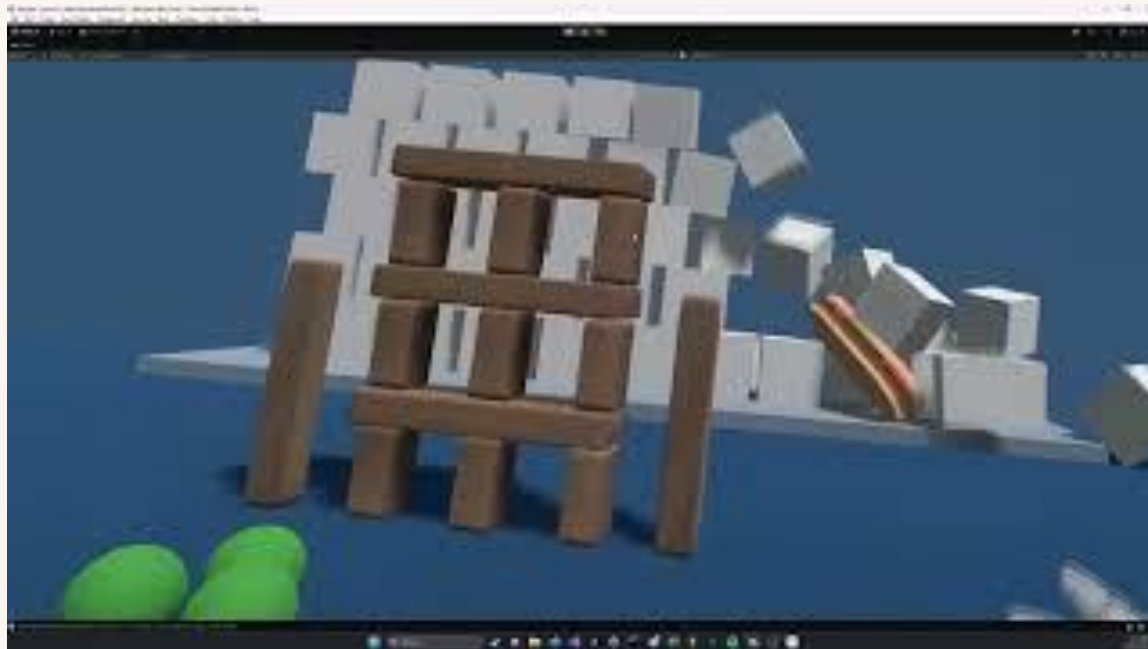
Task memory maximum allocation for container memory reservation

Memory (MiB) bar chart: 0 to 500

Containers [Info](#)

Container name	Image	Private registry	Essential	CPU	Memory hard/soft limit	GPU
UnityGameServer	<a href="#">767397769064.dkr.ecr.us-west-1.amazonaws.com/unitygame-server</a>	-	Yes	25 vCPU	.5 GiB/-	-

# Demo Video



# Thank You

CSU - Fullerton

CPSC 491