

# VRecover

Daniel Currey

Dan Solis

Kaixiang Liu

Jonathan Mai

Jason Chiu

# 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 VRcover
{
    public interface IInitializable
    {
        void Initialize();
    }

    [DisallowMultipleComponent]
    public class Bootstrap : MonoBehaviour
    {
        public static bool isInitialized = false;
        private static GameObject servicesContainer;

        [RuntimeInitializeOnLoadMethod(RuntimeInitializeLoadType.BeforeSceneLoad)]
        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.");
    }

    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.");
}

private static bool RegisterServicesFromPrefabs()
{
    // Add code for registering services from prefabs here
}
```

```
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("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.");
                    continue;
                }

                bool serviceRegistered = false;
                foreach (MonoBehaviour service in services)
                {
                    if (service is IInitializable)
                    {
                        try
                        {
                            Debug.Log($"[Bootstrap] Initializing service: {service.GetType().Name}");
                            IInitializable.Initialize();
                        }
                        catch (Exception ex)
                        {
                            Debug.LogError($"[Bootstrap] Failed to initialize service {service.GetType().Name}: {ex.Message}");
                            continue;
                        }
                    }
                }
            }
        }
    }
}
```

# Service Locator

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

namespace VRcover
{
    public class ServiceLocator
    {
        private static ServiceLocator _instance;
        public static ServiceLocator Instance => _instance ??= new ServiceLocator();

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

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

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

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

        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;
        }

        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 System.Linq;
using UnityEngine.SceneManagement;
using UnityEngine.SceneManagementManagement;
using FishNet.Managing;

namespace VRcover
{
    [System.Serializable]
    public interface IGameManagerService
    {
        [System.Serializable]
        public enum ServerStateType CurrentServerState { get; }

        [System.Serializable]
        public enum ClientStateType CurrentClientState { get; }

        [System.Serializable]
        public enum SessionStateType CurrentSessionState { get; }

        [System.Serializable]
        public enum SceneStateType CurrentSceneState { get; }

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

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

        void RelaunchScene();
        void SwitchScene(string sceneName, bool updateCurrentScene);
        void LoadScene(int sceneNum, bool updateCurrentScene);
        void EndGame();
    }

    [System.Serializable]
    public class NetworkManager : NetworkManagerService : MonoBehaviour, IGameManagerService, IInitializable
    {
        [System.Serializable]
        public enum ServerStateType CurrentServerState { get; private set; }

        [System.Serializable]
        public ClientStateType CurrentClientState { get; private set; }

        [System.Serializable]
        public SessionStateType CurrentSessionState { get; private set; }

        [System.Serializable]
        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 IGameManager _currentServerState;
        private IGameManager _currentClientState;
        private IGameManager _currentSessionState;
        private IGameManager _currentSceneState;

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

```

```

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 NetworkStateHandler _networkStateHandler;

28 references
public NetworkManager NetworkManager => _networkManager;
23 references
public NetworkStateHandler NetworkStateHandler => _networkStateHandler;

2 references
public void Initialize()
{
}

@Unity Message | 0 references
public void Start()
{
    _networkManager = GameObject.FindFirstObjectOfType<NetworkManager>();
    if (_networkManager == null)
    {
        Debug.LogError("GameManagerService: NetworkManager not found in scene!");
        return;
    }

    _networkStateHandler = new NetworkStateHandler(_networkManager);

    QualitySettings.vSyncCount = 6;

    _networkStateSync = FindFirstObjectOfType<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");
}

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

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

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

```

```

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

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

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

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

    // Initialize scene states
    foreach (SceneStateType stateType in Enum.GetValues(typeof(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 enum 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];

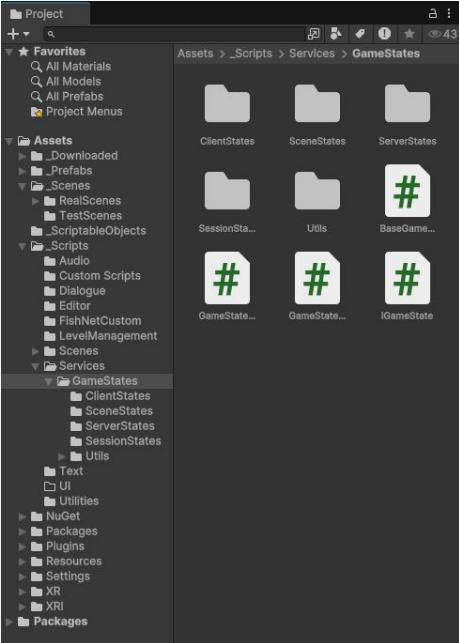
    // Notify listeners for each initial state
    Debug.Log("GameManagerService: Initializing initial state entries");
    _currentServerState.OnEnter(this);
    _currentClientState.OnEnter(this);
    _currentSessionState.OnEnter(this);
    _currentSceneState.OnEnter(this);

    // Notify listeners for each current state
    Debug.Log("GameManagerService: Invoking current state entries");
    OnServerStateChanged?.Invoke(CurrentServerState);
    OnClientStateChanged?.Invoke(CurrentClientState);
    OnSessionStateChanged?.Invoke(CurrentSessionState);
    OnSceneStateChanged?.Invoke(CurrentSceneState);

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

```

# Game States



```
namespace VRcover
{
    // <summary>
    // Represents the state of the AWS Fargate server instance
    // </summary>
    49 references
    public enum ServerStateType
    {
        NONE,                                // Initial state
        DEFAULT,                             // Setup state
        PROVISIONING,                        // AWS Fargate container is being provisioned
        STARTING,                            // Server is starting up
        RUNNING,                             // Server is running and ready for connections
        STOPPING,                            // Server is gracefully shutting down
        STOPPED,                             // Server encountered an error
        SAVING_SESSION_DATA,                 // Saving session data to S3
    }

    // <summary>
    // Represents the client's connection and authentication state
    // </summary>
    // <remarks>
    // Represents the client's connection and authentication state
    // </remarks>
    47 references
    public enum ClientStateType
    {
        NONE,                                // Initial state
        DEFAULT,                             // Setup state
        IN_MAIN_MENU,                         // Client is in the main menu
        AUTHENTICATING,                      // Client is authenticating with AWS Cognito
        CONNECTING,                           // Client is connecting to the server
        CONNECTED,                            // Client is connected to the server
        DISCONNECTING,                        // Client is gracefully disconnecting
        DISCONNECTED,                         // Client encountered an error
        ERROR,                               // Client encountered an error
        WAITING_FOR_SCENE,                   // Waiting for scene to load/sync
    }

    // <summary>
    // Represents the therapy session state
    // </summary>
    // <remarks>
    74 references
    public enum SessionStateType
    {
        NONE,                                // Initial state
        DEFAULT,                             // Setup state
        WAITING_FOR_THERAPIST,               // Waiting for therapist to join
        WAITING_FOR_CLIENT,                  // Waiting for client to join
        PREPARING,                           // Both therapist and client are preparing to start
        IN_PROGRESS,                          // Session is actively running
        PAUSED,                               // Session is temporarily paused
        COMPLETED,                           // Session has ended normally
        ERROR,                               // Session encountered an error
        SAVING,                               // Saving session data
        ANALYZING,                           // Processing session metrics
    }

    // <summary>
    // Represents the VR scene state
    // </summary>
    // <remarks>
    47 references
    public enum SceneStateType
    {
        NONE,                                // Initial state
        DEFAULT,                             // Setup state
        LOADING,                            // Scene is loading
        INITIALIZING,                        // Scene is setting up (post-load)
        ACTIVE,                               // Scene is running
        RUNNING,                            // Therapist is modifying the scene
        TRANSITIONING,                      // Transitioning between scenes
        COMPLETED,                           // Scene objectives completed
        ERROR,                               // Scene encountered an error
        SYNCHRONIZING,                       // Syncing scene state between clients
    }
}
```

# AWS Login Manager

```
C:\> Users > Dan > Documents > Git Desktop Projects > vrecover > Assets > Custom Scripts > LoginManager.cs
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 TMPro;
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(AttemptingLogin()));
39     }
40
41     private IEnumerator AttemptingLogin()
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         {
50             errorText.text = "Both fields are required";
51             return;
52         }
53
54         // Create a CognitoUser object
55         CognitoUser user = new CognitoUser(_userPool, username);
56
57         // Set the password
58         user.SetPassword(password);
59
60         // Perform the sign-in
61         var result = user.SignIn();
62
63         if (result.SignedIn)
64         {
65             // User signed in successfully
66             // You can now update the UI to show the user is logged in
67             // For example, you can change the button text to "Logout" or update a UI element
68             // ...
69         }
70         else
71         {
72             // User failed to sign in
73             errorText.text = "Sign-in failed: " + result.Exception.Message;
74         }
75     }
76 }
```

# AWS User Custom Attribute

The screenshot shows two AWS IAM users: 'dev' and 'client'. Both users have a custom attribute 'custom:role' with the value 'therapist' for 'dev' and 'client' for 'client'. The 'client' user was created on November 25, 2024, and updated on December 9, 2024. The 'dev' user was created on November 18, 2024, and updated on November 25, 2024.

**User: dev**

**User information**

- User ID (Sub): 91fb540-6081-7049-c4da-efc661fc8953
- Alias attributes used to sign in: User name
- MFA setting: MFA inactive
- MFA methods: -

**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

**User attributes (2)**

View and edit this user's attributes.

Attribute name	Value	Type
custom:role	therapist	Optional
sub	91fb540-6081-7049-c4da-efc661fc8953	Required

**Group memberships (0)**

View and edit this user's group memberships.

Group name	Description	Group created time

**User: client**

**User information**

- User ID (Sub): f12bd510-f081-70fc-e5f0-46628277e78f
- Alias attributes used to sign in: User name
- MFA setting: MFA inactive
- MFA methods: -

**Account status**: Enabled

**Confirmation status**: Confirmed

**Created time**: November 25, 2024 at 10:13 PST

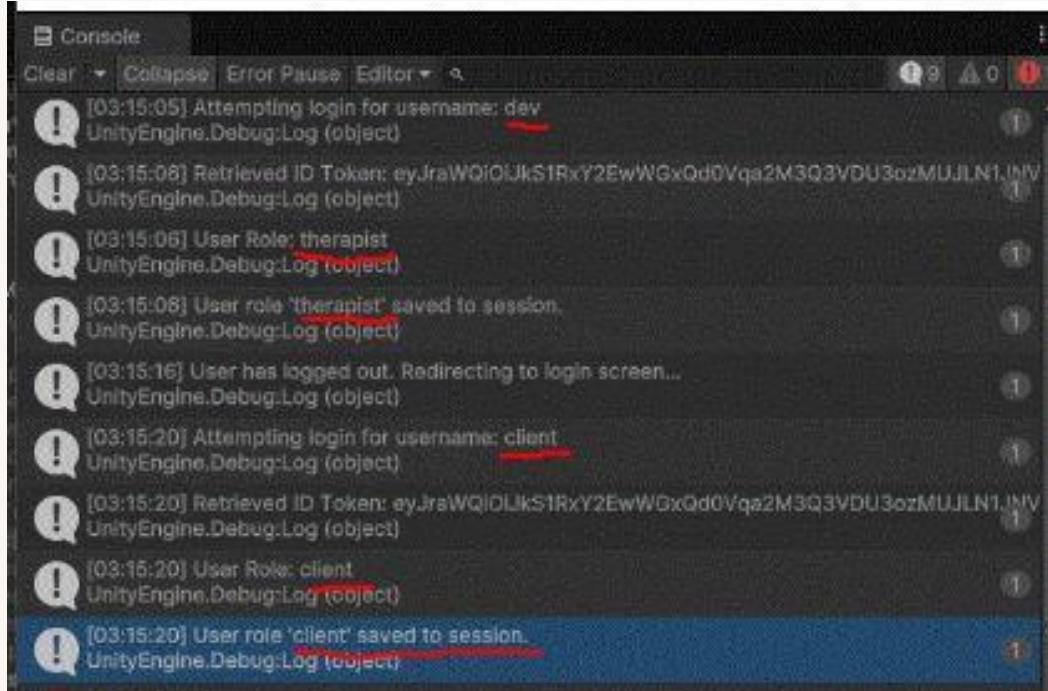
**Last updated time**: December 9, 2024 at 02:32 PST

**User attributes (2)**

View and edit this user's attributes.

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

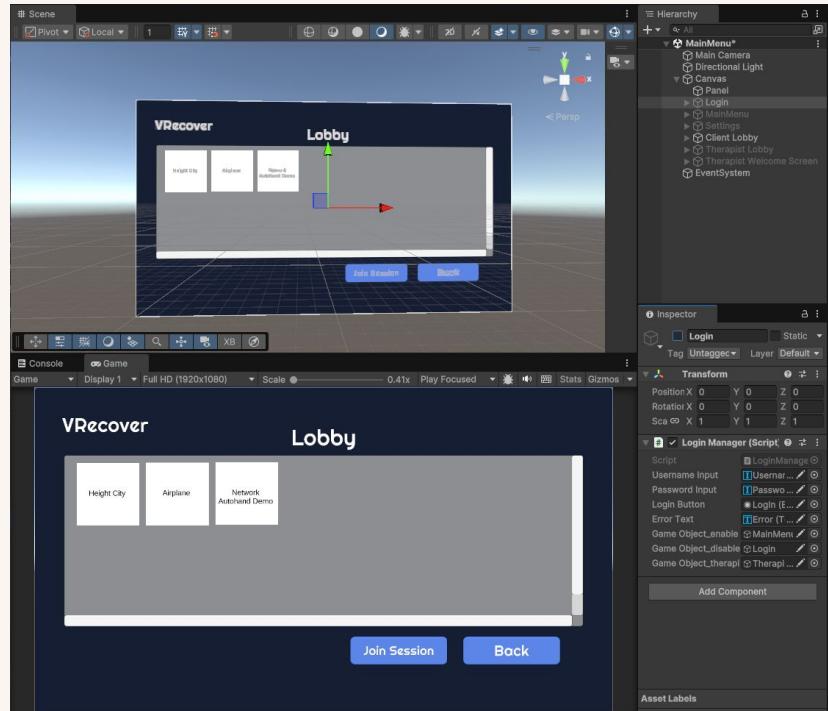
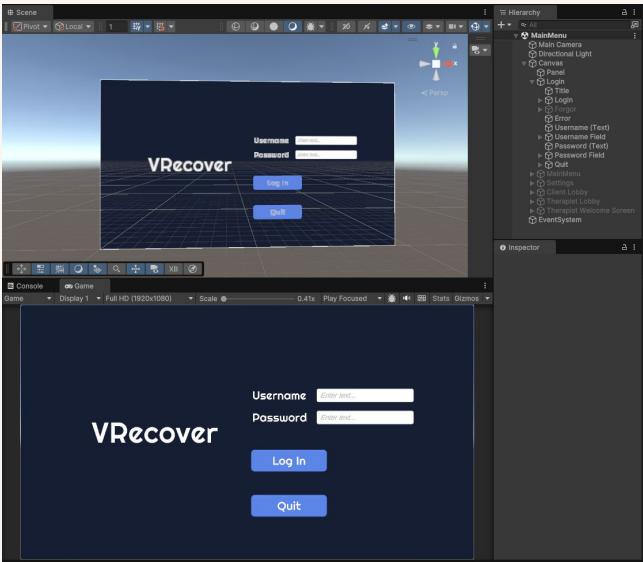
# Attribute Reading



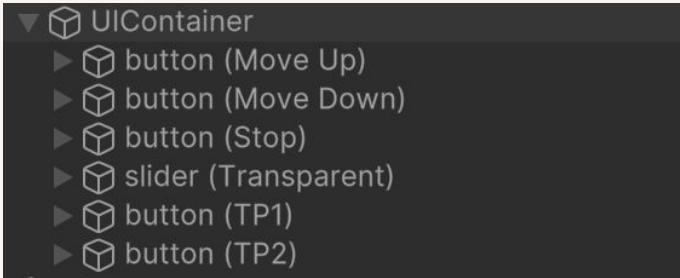
The screenshot shows a Unity Editor Console window with the following log entries:

- [03:15:05] Attempting login for username: dev  
UnityEngine.Debug:Log (object)
- [03:15:08] Retrieved ID Token: eyJraWQ1QJKs1RxY2EwWGxQd0Vqa2M3Q3VDU3ozMUJLN1JMV  
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: eyJraWQ1QJKs1RxY2EwWGxQd0Vqa2M3Q3VDU3ozMUJLN1JMV  
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)

# Menu



# Serialized UI



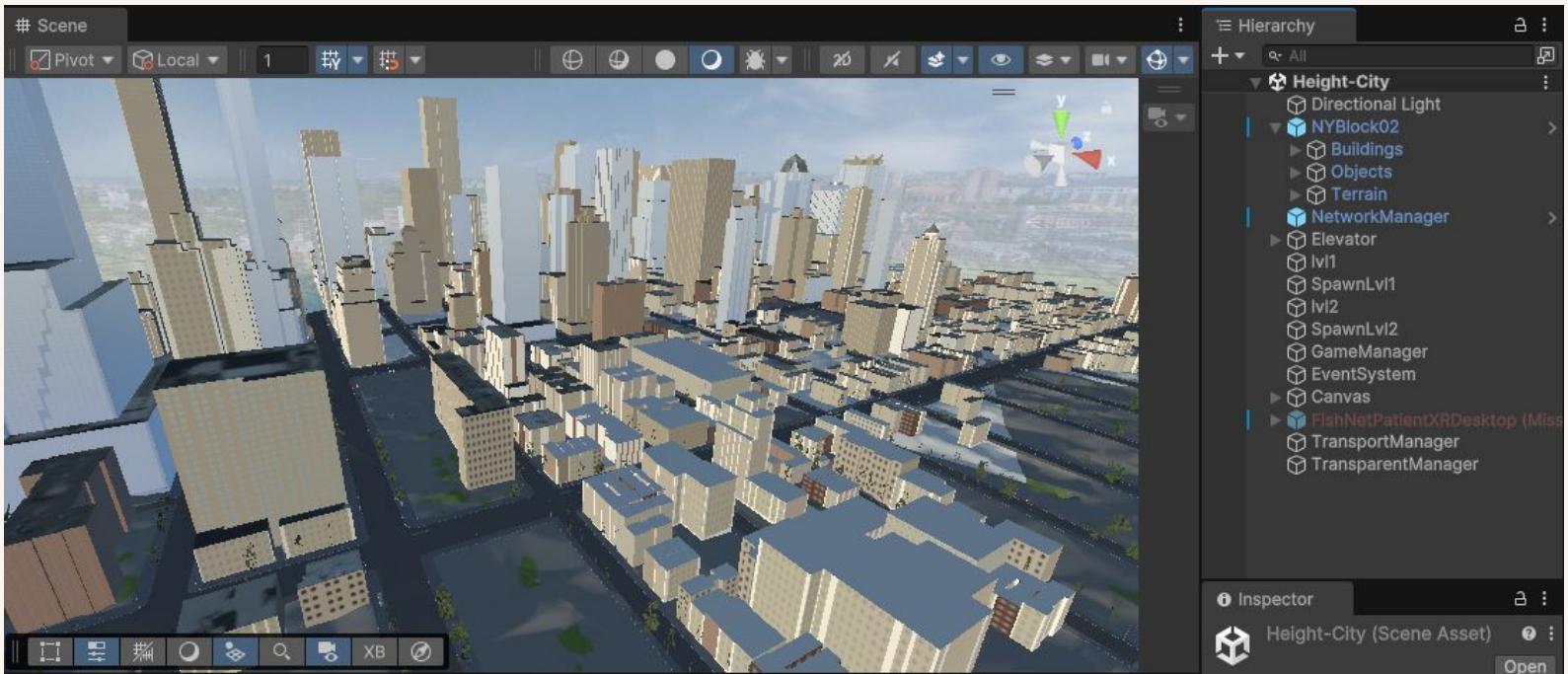
```
1  {
2    "scene": {
3      "name": "jon-test-scene",
4      "elements": [
5        {
6          "type": "button",
7          "label": "Move Up",
8          "targetObject": "Elevator",
9          "method": "MoveElevatorUp"
10        },
11        {
12          "type": "button",
13          "label": "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  }
```

# DynamicUIManager

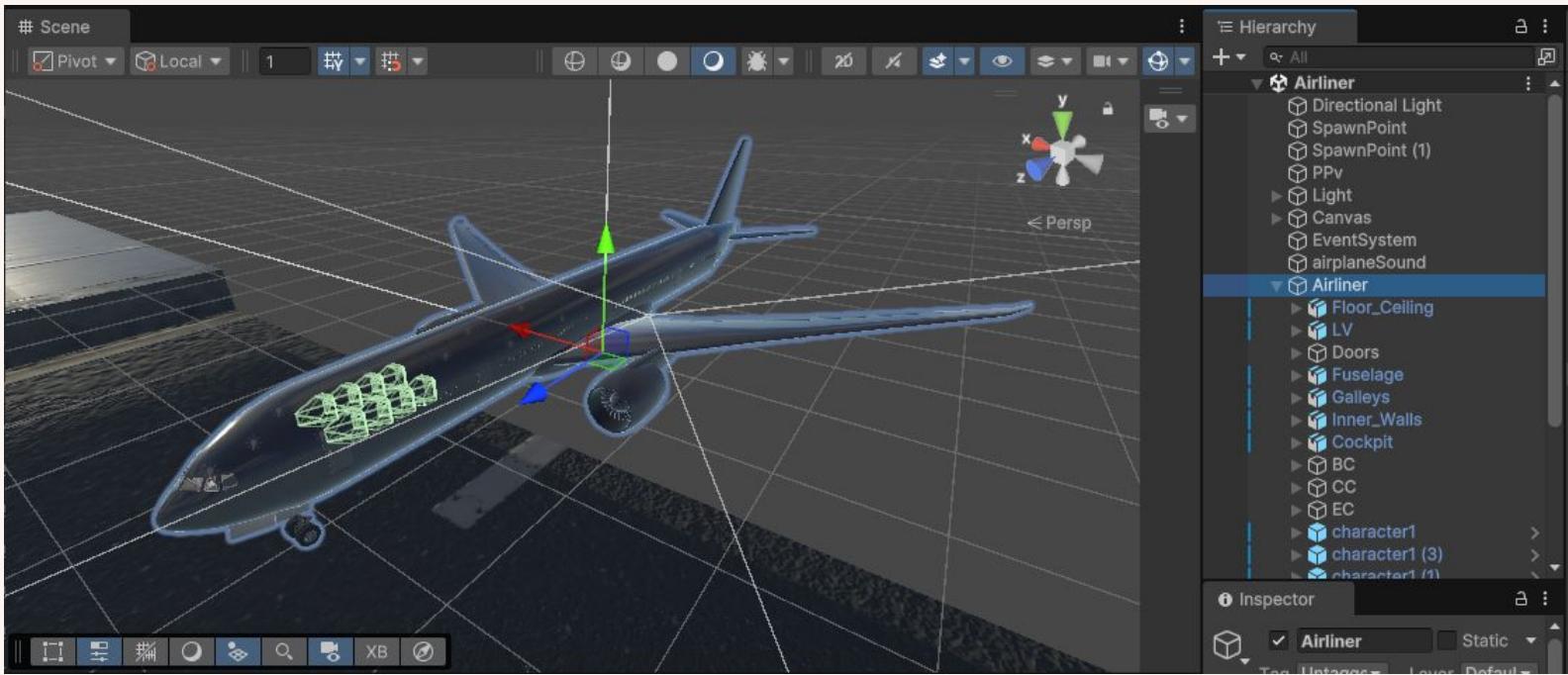
```
1  using UnityEngine;
2  using Newtonsoft.Json;
3  using System;
4  using System.Collections;
5  using UnityEngine;
6  using UnityEngine.UI;
7  using System.Reflection;
8
9  public class DynamicUIManager : MonoBehaviour
10 {
11
12     using UnityEngine;
13     using Newtonsoft.Json;
14     using System;
15     using System.Collections;
16     using UnityEngine;
17     using UnityEngine.UI;
18     using System.Reflection;
19
20     [Serializable]
21     private GameObject sliderPrefab;
22     [Serializable]
23     private GameObject togglePrefab;
24     [Serializable]
25     private GameObject buttonPrefab;
26
27     [Serializable]
28     private Transform canvas;
29
30     [Serializable]
31     private Transform uiContainer;
32
33     void Start()
34     {
35         SetupContainer();
36
37         SceneMetadata sceneMetadata = LoadMetadata(metadataFileName);
38         if (sceneMetadata == null)
39         {
40             Debug.LogError("Failed to load metadata.");
41             return;
42         }
43
44         if (string.Equals(sceneMetadata.name, UnityEngine.SceneManagement.SceneManager.GetActiveScene().name, StringComparison.OrdinalIgnoreCase))
45         {
46             Debug.LogWarning($"Metadata is for scene '{sceneMetadata.name}', but current scene is '{UnityEngine.SceneManagement.SceneManager.GetActiveScene().name}'.");
47             return;
48         }
49
50         foreach (var element in sceneMetadata.elements)
51         {
52             GameObject targetObject = GameObject.Find(element.targetObject);
53             if (targetObject == null)
54             {
55                 Debug.LogWarning($"target object '{element.targetObject}' not found in the scene.");
56                 continue;
57             }
58
59             CreateUIElement(element, targetObject);
60         }
61     }
62
63     private void SetupContainer()
64     {
65         if (uiContainer == null)
66         {
67             GameObject containerObject = new GameObject("UIContainer");
68             containerObject.transform.SetParent(canvas, false);
69
70             RectTransform rectTransform = containerObject.AddComponent<RectTransform>();
71             rectTransform.anchorMin = new Vector2(1, 1);
72             rectTransform.anchorMax = new Vector2(0, 0);
73             rectTransform.offsetMax = Vector2.zero;
74
75             var layoutGroup = containerObject.AddComponent<VerticalLayoutGroup>();
76             layoutGroup.spacing = 2;
77             layoutGroup.childAlignment = TextAnchor.MiddleCenter;
78
79             var contentFitter = containerObject.AddComponent<ContentSizeFitter>();
80             contentFitter.verticalFit = ContentSizeFitter.FitMode.PreferredSize;
81
82             uiContainer = containerObject.transform;
83         }
84     }
85
86     private void LoadMetadata(string fileName)
87     {
88         TextAsset jsonFile = Resources.Load<TextAsset>(fileName);
89         if (jsonFile == null)
90         {
91             Debug.LogError($"Metadata file '{fileName}' not found in Resources folder.");
92             return null;
93         }
94     }
95 }
```

```
95
96     private void CreateUIElement(UIElement element, GameObject targetObject)
97     {
98         GameObject uiElement = null;
99
100         if (string.Equals(element.type, "slider", StringComparison.OrdinalIgnoreCase))
101         {
102             uiElement = Instantiate(sliderPrefab);
103             ConfigureSlider(uiElement, element, targetObject);
104         }
105         else if (string.Equals(element.type, "toggle", StringComparison.OrdinalIgnoreCase))
106         {
107             uiElement = Instantiate(togglePrefab);
108             ConfigureToggle(uiElement, element, targetObject);
109         }
110         else if (string.Equals(element.type, "button", StringComparison.OrdinalIgnoreCase))
111         {
112             uiElement = Instantiate(buttonPrefab);
113             ConfigureButton(uiElement, element, targetObject);
114         }
115
116         if (uiElement != null)
117         {
118             uiElement.transform.SetParent(uiContainer, false);
119             uiElement.name = $"{element.type} {element.label} {targetObject.name}";
120             label.text = element.label;
121         }
122     }
123
124     private void ConfigureSlider(GameObject sliderObj, UIElement element, GameObject targetObject)
125     {
126         var label = sliderObj.transform.Find("Label").GetComponent<UIPro.TextMeshProUGUI>();
127         label.text = element.label;
128
129         var slider = sliderObj.GetComponentInChildren<UnityEngine.UI.Slider>();
130
131         object currentValueObj = PropertyUtility.GetValue(targetObject, element.property);
132         if (currentValueObj is float currentValue)
133         {
134             float range = element.max - element.min;
135             slider.minValue = currentValue - range / 2f;
136             slider.maxValue = currentValue + range / 2f;
137             slider.value = currentValue;
138
139             slider.onValueChange.AddListener(value =>
140             {
141                 PropertyUtility.SetValue(targetObject, element.property, value);
142             });
143         }
144     }
145
146     private void ConfigureToggle(GameObject toggleObj, UIElement element, GameObject targetObject)
147     {
148         var label = toggleObj.GetComponentInChildren<UIPro.TextMeshProUGUI>();
149         label.text = element.label;
150
151         var toggle = toggleObj.GetComponentInChildren<UnityEngine.UI.Toggle>();
152         toggle.isOn = Convert.ToBoolean(element.defaultValue);
153
154         toggle.onValueChange.AddListener(isOn =>
155         {
156             PropertyUtility.SetValue(targetObject, element.property, isOn);
157         });
158     }
159
160     private void ConfigureButton(GameObject buttonObj, UIElement element, GameObject targetObject)
161     {
162         var label = buttonObj.GetComponentInChildren<UIPro.TextMeshProUGUI>();
163         label.text = element.label;
164
165         var button = buttonObj.GetComponentInChildren<UnityEngine.UI.Button>();
166         button.onClick.AddListener(() =>
167         {
168             // Look for any MonoBehaviour scripts on the target object
169             var scripts = targetObject.GetComponents<MonoBehaviour>();
170             if (scripts.Length == 0)
171             {
172                 Debug.LogError($"No MonoBehaviour scripts found on {targetObject.name}.");
173             }
174         });
175
176         // Loop through each script to find the specified method
177         foreach (var script in scripts)
178         {
179             if (script.name == method)
180             {
181                 return;
182             }
183         }
184     }
185 }
```

# 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

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

# GitHub Actions Runs

Code / Issues Full requests Actions Projects Security Insights Settings

All workflows Showing runs from all workflows

13 workflow runs

Event	Status	Branch	Actor
Merge pull request #28 from spicy/dev	in progress	main	16 hours ago
Resolved all .meta file conflicts from main	in progress	main	yesterday
Merge pull request #17 from spicy/dev	in progress	main	3 hours ago
Resolved all .meta file conflicts from main	in progress	main	1m 40s
Merge pull request #17 from spicy/dev	in progress	main	1m 36s
Merge pull request #17 from spicy/dev	in progress	main	last week
Merge pull request #17 from spicy/dev	in progress	main	2 weeks ago
Merge pull request #16 from spicy/dev	in progress	main	2 weeks ago
Merge pull request #15 from spicy/dev	in progress	main	2 weeks ago
Merge pull request #14 from spicy/dev	in progress	main	2 weeks ago
Merge pull request #13 from spicy/dev	in progress	main	1m 56s
Merge pull request #13 from spicy/dev	in progress	main	1m 56s
Merge pull request #10 from spicy/dev	in progress	main	last month
Merge pull request #9 from spicy/dev	in progress	main	last month
Merge pull request #6 from spicy/dev	in progress	main	last month

Build and Deploy Unity Server to AWS Fargate

Resolved all .meta file conflicts from main #12

Summary

Jobs

deploy

Run details

Usage

Workflow file

Annotations 3 warnings

Search logs

Success yesterday in 1m 51s

Set up job 1s

Checkout Code 8s

Configure AWS Credentials 8s

Log in to Amazon ECR 1s

Build Docker Image 59s

Push Docker Image to ECR 35s

Register ECS Task Definition 4s

Update ECS Service 8s

Post Log in to Amazon ECR 6s

Post Configure AWS Credentials 8s

Post Checkout Code 8s

Complete job 8s

# AWS Repository

## Private repositories (1)

Search by repository substring

Repository name

▲ URI

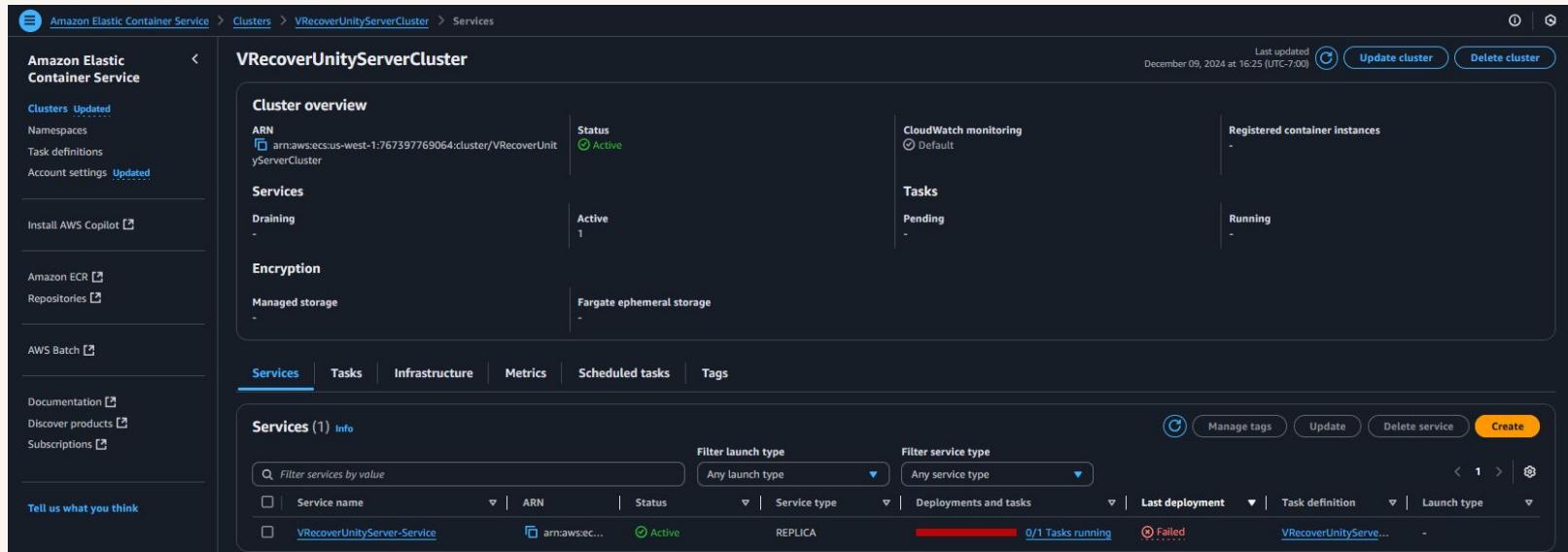
unity-vr-therapy-server

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

The screenshot shows the AWS ECR repository details page for the 'unity-vr-therapy-server' repository. The repository has 1 private image. The image details are as follows:

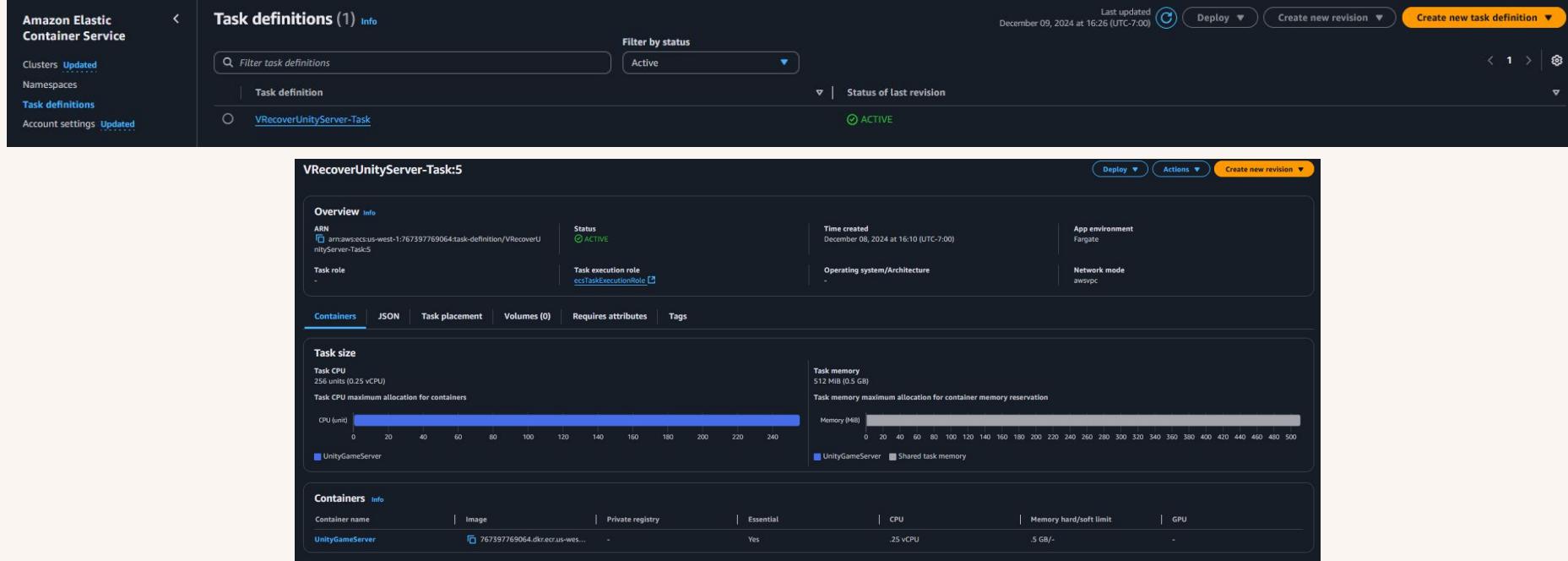
- Image**
- Details**
  - Image tags: latest
  - URI: 767397769064.dkr.ecr.us-west-1.amazonaws.com/unity-vr-therapy-server/latest
  - Digest: sha256:d7e09e6a5ca9dc18c0908243ea00e6aed2c89287ee1824b7bf49565a15ccdc11
- General information**
  - Artifact type: Image
  - Repository: unity-vr-therapy-server
  - Pushed at: December 08, 2024, 16:10:57 (UTC-07)
  - Size (MB): 639.96
- Scanning and vulnerabilities**
  - Status: Scan not found
  - Scan: Scan
- Referrers**
  - Info
  - Referrer digest
  - Type
  - Pushed at
  - No referrers
  - This artifact does not have any referrers
- Replication status**
  - Info
  - Select status source: Querries for replication status based on selection of a specific tag or by the image digest
  - digest
  - Filter status: Filter status
  - Target account: ▲ Region: ▼ Status: ▼ Error: ▼
  - No replication status
  - Queried tag or digest does not have replication statuses

# AWS Fargate, Clusters



The screenshot shows the AWS Elastic Container Service (ECS) Cluster Overview page for the cluster 'VRecoverUnityServerCluster'. The page is divided into several sections: Cluster overview, Services, Encryption, CloudWatch monitoring, Tasks, and Registered container instances. The 'Services' section is currently selected. It displays a table with one service entry: 'VRecoverUnityServer-Service' (arn:aws:ecs:us-west-1:1767397769064:cluster/VRecoverUnityServerCluster). The service is active with 1 task running, and the last deployment failed. The 'Encryption' section shows that managed storage is using Fargate ephemeral storage. The 'CloudWatch monitoring' section shows default settings. The 'Tasks' section shows pending tasks. The 'Registered container instances' section is currently empty. At the bottom, there are tabs for Services, Tasks, Infrastructure, Metrics, Scheduled tasks, and Tags, with 'Services' being the active tab. There are also buttons for Manage tags, Update, Delete service, and Create.

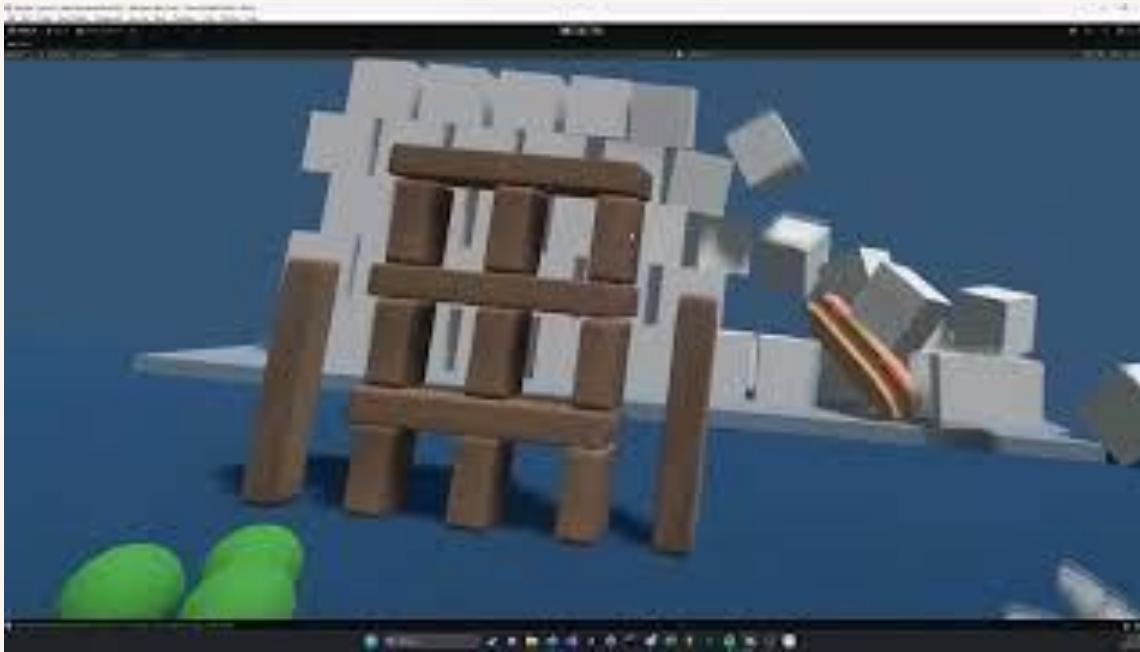
# AWS Fargate Task Definitions



The screenshot shows the AWS Elastic Container Service (ECS) Task Definitions page. The main header displays "Task definitions (1) Info" and "Last updated December 09, 2024 at 16:26 (UTC-7:00)". The left sidebar includes links for Clusters, Namespaces, Task definitions, and Account settings. The main content area shows a single task definition named "VRecoverUnityServer-Task" with the status "ACTIVE". The "Overview" tab is selected, showing details like ARN, Status (ACTIVE), Time created (December 08, 2024 at 16:10 (UTC-7:00)), and App environment (Fargate). The "Containers" tab is also visible. Below the overview, there are sections for "Task size" (Task CPU: 256 units (0.25 vCPU), Task memory: 512 MiB (0.5 GB)) and "Containers" (UnityGameServer). The "Containers" table includes columns for Container name, Image, Private registry, Essential, CPU, Memory hard/soft limit, and GPU.

Container name	Image	Private registry	Essential	CPU	Memory hard/soft limit	GPU
UnityGameServer	767397769064.dkr.ecr.us-west-2.amazonaws.com/unitygameserver:latest	-	Yes	.25 vCPU	.5 GB/-	-

# Demo Video



# Thank You

CSU - Fullerton

CPSC 491