

ReLayer UI Updates with AWS S3 and CloudFront

Real AWS Console screens in English (US). Identifying regions are cropped out; example forms were not submitted. No AWS resources were created or changed.

Deliver compatible UI changes without rebuilding your Unity application.

Use this workflow for menus, dashboards, training tools, visualizations, or other interfaces.

All names and addresses below are placeholders; replace them with your own.

1. Choose a delivery mode

Mode	Delivery	Offline behavior
Hosted	Opens HTML, JavaScript, and CSS over HTTPS; checks for updates automatically	Falls back to the built-in UI
Addressables	Downloads a Unity AssetBundle containing a verified UI package	Can reuse the last verified local installation

Ship an application containing the ReLayer loader and a built-in fallback once before publishing UI updates.

This updates web UI/content, not C# or native code; WebGL hot updates are not officially supported.

AWS services may incur charges. Start with a test environment and set a budget alert.

2. Set up S3 and CloudFront once

1. In **S3 > Create bucket**, choose a unique name and your preferred AWS Region.
2. Keep **Block Public Access** enabled and **Object Ownership > ACLs disabled**.
3. Use **SSE-S3** encryption; enable bucket versioning for recovery, then create the bucket. Leave static website hosting disabled.

1a Object ownership

AWS CONSOLE / S3

- ☒ **ACLs disabled (recommended)**
All objects in this bucket are owned by this account. Access to this bucket and its objects is specified using only policies.

Object Ownership
Bucket owner enforced

S3 > Create bucket > Object Ownership: retain ACLs disabled and Bucket owner enforced.

1b Block public access

AWS CONSOLE / S3

- ☒ **Block all public access**
Turning this setting on is the same as turning on all four settings below. Each of the following settings are independent of one another.

S3 > Create bucket: retain Block all public access. The four dependent controls remain enabled.

2 Bucket versioning

AWS CONSOLE / S3

Bucket Versioning

- ☐ Disable
☒ **Enable**

Bucket Versioning: Enable is recommended for recovery. This example form is unsaved.

3 Default encryption

AWS CONSOLE / S3

- ☒ **Server-side encryption with Amazon S3 managed keys (SSE-S3)**
☐ Server-side encryption with AWS Key Management Service keys (SSE-KMS)
☐ Dual-layer server-side encryption with AWS Key Management Service keys (DSSE-KMS)

Default encryption: use SSE-S3 for this walkthrough. Leave static website hosting disabled.

Connect CloudFront to S3

4. Create a CloudFront distribution; review the pricing plan and choose **Single website or app**, then a regular S3 origin, not its website endpoint.
5. Leave **Origin path** empty; attach **Origin access control (OAC)** with **Sign requests**.
6. Apply CloudFront's suggested S3 read policy, restricted to your distribution ARN; the new wizard can apply it with **Allow private S3 bucket access to CloudFront**. Preserve unrelated statements.

4 S3 origin and path

AWS CONSOLE / CLOUDFRONT

Origin

S3 origin

Choose an AWS origin, or enter your origin's domain name. [Learn more](#)

your-ui-bucket.s3.us-east-1.amazonaws.com

Browse S3

Origin path - optional

The directory path within your origin where your content is stored. [Learn more](#)

/path

CloudFront > Create distribution > Specify origin: regular S3 origin; Origin path stays empty.

5 Origin access control

AWS CONSOLE / CLOUDFRONT

Create new OAC

Name

The name must be unique. Valid characters: letters, numbers and most special characters. Use up to 64 characters.

Enter name

Description - optional

The description can have up to 256 characters.

Enter description

Signing behavior

- ☐ Do not sign requests
- ☒ Sign requests (recommended)

- ☐ Do not override authorization header

Do not sign if incoming request has authorization header.

Origin type

S3

The origin type must be the same type as origin domain.

- ☐ Use SigV4a signing protocol

Enable SigV4a signing for origins that require asymmetric signing.

Cancel

Create

Origin access > Create control setting: Sign requests, S3. Leave Do not override authorization header unchecked.

Configure HTTPS and caching

7. Under **Customize cache settings**, select **HTTPS only** and **GET, HEAD**. Use the generated CloudFront domain; a custom domain is optional.

8. Wait for distribution deployment, then configure these cache behaviors before the default rule:

Path pattern	Cache policy
relayer/hosted/release.json	CachingDisabled
relayer/addressables/*/catalog*	CachingDisabled
Default *	CachingOptimized

Omit the rule for a mode you do not use; adapt catalog patterns if your build uses different filenames.

6 Update path pattern

AWS CONSOLE / CLOUDFRONT

Settings

Path pattern [Info](#)

Origin and origin groups

Compress objects automatically [Info](#)

☐ No
☒ Yes

Distribution > Behaviors > Create behavior: enter the pointer path; choose your S3 origin.

7 Pointer cache policy

AWS CONSOLE / CLOUDFRONT

Cache key and origin requests

We recommend using a cache policy and origin request policy to control the cache key and origin requests.

☒ Cache policy and origin request policy (recommended)
☐ Legacy cache settings

Cache policy

Choose an existing cache policy or create a new one.

Policy with caching disabled

[Create cache policy](#) [View policy](#)

The pointer behavior uses CachingDisabled; keep immutable assets on the default CachingOptimized behavior.

HTTPS and public delivery

Do not rewrite missing files or 403/404 responses to an HTML page with status 200.

8

HTTPS and request methods

AWS CONSOLE / CLOUDFRONT

Viewer protocol policy

- ☐ HTTP and HTTPS
- ☐ Redirect HTTP to HTTPS
- ☒ HTTPS only

Allowed HTTP methods

- ☒ GET, HEAD
- ☐ GET, HEAD, OPTIONS
- ☐ GET, HEAD, OPTIONS, PUT, POST, PATCH, DELETE

Use HTTPS only and GET, HEAD. No distribution, origin, or behavior was saved during capture.

The S3 bucket is private, but files served through this CloudFront configuration are public.

Never place AWS credentials or application secrets in UI files.

See [AWS OAC setup](#) and [cache policies](#).

3. Build and publish a Hosted UI

1. Select **ReLayer Initializer > Player UI Delivery > UI source > Hosted**.
2. Enter `https://d111example.cloudfront.net/relayer/hosted/` as **Hosting directory URL**.
3. Click **Build UI Update** and use the output folder that opens.
4. Upload the new `releases/<version>/` folder first, preserving filenames and directories.
5. Verify the uploaded files, then publish the top-level `release.json` last.

The URL must be an HTTPS directory, not an S3 console link, `s3://` path, or HTML filename.

ReLayer generates versions and manifests; edit your source and rebuild instead of modifying generated files.

The default profile's output is `.relayer/delivery/default/Hosted`; other profiles use different folders.

Output	S3 destination	Cache-Control
<code>releases/<version>/...</code>	<code>relayer/hosted/releases/<version>/...</code>	<code>public,max-age=31536000,immutable</code>
<code>Top-level release.json</code>	<code>relayer/hosted/release.json</code>	<code>no-cache,max-age=0</code>

Keep previous releases. Do not overwrite a version with different content or delete older releases during upload.

Use correct MIME types: HTML `text/html`, JS `text/javascript`, CSS `text/css`, JSON `application/json`.

Disable service-worker interception for this deployment; keep the generated bootstrap and security policy.

Optional PowerShell upload workflow

Run from your Unity project root with Python 3.10+ and an AWS CLI v2 supporting `--no-overwrite`.

Upload options are documented in the [AWS CLI reference](#).

Configure an authorized, least-privilege upload profile first; [AWS CLI SSO setup](#) is one option.

Replace these four values; the upload commands write to your selected bucket.

```
$AwsProfile = 'ui-publisher'
$UiBucket = 'your-ui-bucket'
$CdnBase = 'https://d111example.cloudfront.net'
$UiBuild = '.relayer/delivery/default/Hosted'
$HostedBase = "$CdnBase/relayer/hosted/"
$HostedS3 = "s3://$UiBucket/relayer/hosted"
```

A. Verify locally, upload the version, and verify it over HTTPS:

```
python Assets/ReLayer/Core/Build/verify-ui-update.py `
  --root $UiBuild
if ($LASTEXITCODE -ne 0) { throw 'Local verification failed.' }
$Manifest = Get-Content -LiteralPath (Join-Path $UiBuild 'release.json') -Raw | ConvertFrom-Json
$Version = $Manifest.version
$VersionDirectory = Join-Path (Join-Path $UiBuild 'releases') $Version
aws s3 cp $VersionDirectory "$HostedS3/releases/$Version/" `
  --recursive `
  --no-follow-symlinks `
  --no-overwrite `
  --cache-control 'public,max-age=31536000,immutable' `
  --profile $AwsProfile
if ($LASTEXITCODE -ne 0) { throw 'Upload failed; keep the current pointer.' }
python Assets/ReLayer/Core/Build/verify-ui-update.py `
  --root $UiBuild `
  --remote-base $HostedBase
if ($LASTEXITCODE -ne 0) { throw 'Remote verification failed; do not publish.' }
```

Require `remoteFilesVerified: true`; inspect actual HTML/JS/CSS response MIME types before continuing.

The verifier checks file sizes and SHA-256, not MIME types, rendering, or Unity bridge behavior.

B. Only after those checks pass, publish the pointer and verify it:

```
aws s3 cp (Join-Path $UiBuild 'release.json') "$HostedS3/release.json" `
  --content-type 'application/json' `
  --cache-control 'no-cache,max-age=0' `
  --profile $AwsProfile
if ($LASTEXITCODE -ne 0) { throw 'Pointer upload failed; inspect deployed state.' }
python Assets/ReLayer/Core/Build/verify-ui-update.py `
  --root $UiBuild `
  --remote-base $HostedBase `
  --require-pointer
if ($LASTEXITCODE -ne 0) { throw 'Pointer verification failed; inspect or roll back.' }
```

Require `publishedPointerVerified: true`. Run only one deployment per channel at a time.

If using the S3 console instead, follow the same file-first, verification, pointer-last sequence.

4. Apply updates safely

Hosted checks at startup and every 30 seconds while foregrounded; discovery does not refresh the current screen.

A prepared update applies when your application explicitly hides and reopens the entire ReLayer view using `SetVisibility(false/true)`.

For an always-visible interface, call this at a loading transition or another application-owned safe point:

```
await initializer.ApplyPreparedPlayerOverlayAsync();
```

Hiding a React panel, changing routes, disabling a `GameObject`, or restoring app focus is not this safe point.

Without a safe point, the current UI remains until the next launch.

Use `RestorePlayerOverlayStateAsync` to resend application state before revealing the replacement.

React component state is not preserved; restore snapshots, never replay purchases or other commands.

See the state restoration example (see `PlayerUiUpdates.md` in the package).

5. Use Addressables for downloaded, offline-capable UI

Install an Editor-compatible Addressables version; create settings under **Window > Asset Management > Addressables > Groups**.

Windows tested pairs: Unity 2022.3.62f2 / Addressables 1.21.19; Unity 6000.5.2f1 / Addressables 2.10.3. Version 1.21.19 cannot compile on Unity 6000.5.

Use a test profile or existing workflow; preserve production settings.

Setting	Example
Remote.BuildPath	ServerData/[BuildTarget]
Remote.LoadPath	https://d111example.cloudfront.net/relayer/addressables/app-001/[BuildTarget]
Build Remote Catalog	Enabled; catalog paths use the same remote profile variables
ReLayer UI source	Addressables
Update control	Automatic, or Project when your application already updates catalogs

1. Click **Set up UI Addressables** to create the dedicated UI group and key.
2. Click **Build UI Update**; approve **Create new baseline** only for a new application build.
3. Preserve Assets/ReLayerGenerated/Delivery/<profile>/Baselines/ and **Advanced > Addressables Content State Path**.
4. Publish the actual Remote.BuildPath output to the matching S3 prefix, keeping its platform and filenames.
5. Upload new bundles first; verify their downloaded sizes and SHA-256 against local files.
6. Upload and verify the catalog (.json or .bin), then its hash. Never upload private content-state files.
7. Use immutable caching for hash-named bundles and no-cache,max-age=0 for catalog/hash files.
8. Build and distribute the application once using that exact content baseline.

For example, ServerData/Android maps to relayer/addressables/app-001/Android/; build each platform separately.

Keep this baseline URL stable for updates. Upload bundles/catalog/hash, not just the generated overlay.bytes.

Prevent an unintended second content build: use **Do not Build Addressables content on Player Build** for this manual workflow.

If CI owns builds, retain its workflow and use ReLayer's build integration APIs (see PlayerUiUpdates.md in the package).

For later UI changes, retain the deployed player's Editor/Addressables versions and original baseline; click **Build UI Update** and repeat publishing.

Catalog and hash are separate objects: publishing the hash last is not an atomic switch; test failures and retry behavior.

The Hosted verifier above is not an AssetBundle deployment verifier.

See [Unity remote content](#) and [content updates](#).

With **Automatic**, Addressables performs its normal startup catalog check; startup catalog updates must be enabled.

With **Project**, call this after your existing catalog updater succeeds:

```
await initializer.InitializePlayerOverlayAsync();
```

For another in-session update, update the catalog first, call PreparePlayerOverlayAsync(), then apply at a safe point.

ReLayer does not duplicate global catalog updates or manage unrelated content lifetimes.

6. Optional production improvements

- Keep startup, download, and recovery screens built in; separate their output from the remote interface.
- Put large media in separate content packs so UI-only edits need not redownload unchanged media.
- Use Addressables byte-progress APIs for download displays; ReLayer's status progress represents processing phases.
- Use `PlayerOverlayRestoreVersion` to restore state and media belonging to the version actually being activated.
- Signed release pointers, mandatory updates, deployment approvals, and whole-application recovery are project-owned extensions.
- Multiple publishers should use [S3 conditional writes](#) to prevent conflicting updates.

These are optional patterns, not additional requirements for the basic Hosted tutorial.

7. Verify updates and recovery

- Run v1, publish a visible v2 change, and confirm detection without rebuilding or restarting the application.
- Confirm the current screen remains unchanged until the safe point, then test rendering and Unity bridge messages.
- Test a failed download, missing file, state restoration, and offline restart on your target devices.
- Hosted offline recovery uses the built-in UI; Addressables can reuse a verified installation if app data remains.
- Roll back Hosted by verifying a retained release and republishing its top-level `release.json`.
- Roll back Addressables with the matching earlier catalog/hash and retained bundles from the same application baseline.
- If a failed version is quarantined, publish a corrected version or use **Retry failed version** after fixing its cause.

For 403 errors, check OAC, bucket policy, and object paths; do not make the bucket public.

For stale updates, check cache-behavior matching; invalidate only affected paths when correcting old cache mistakes.

For blank UI, check MIME types, missing files, and unwanted HTML error-page rewrites.

For blocked external resources, configure allowed resource origins and server CORS without weakening bridge security.

A successful upload is not proof of successful runtime activation; validate the actual deployed application.