# Skybox AI Generator by Blockade Labs

Create stunning AI-generated skybox assets within Unity for use as skyboxes, HDRI style lighting, and world meshes. Speed up your prototyping or use in your existing creative workflow!

## Prerequisites

In order to use this package you need to provide an API key from Blockade Labs in the API section. Get one at https://api.blockadelabs.com.

Unity 2020.3 LTS or higher.

## Installation

### Install from the Unity Asset Store

- Add this package to your assets from the Unity Asset Store
- Go to `Window > Package Manager`
- Change the project scope to `My Assets`
- Find `Blockade Labs SDK` and click `Install`

### Install with OpenUPM

- CLI:
  - `openupm add com.blockadelabs.sdk`
- Registry:
  - Name: `OpenUPM`
  - URL: `https://package.openupm.com`
  - Scope: `com.blockadelabs`

### Install as a git package

- Go to `Window > Package Manager`
- `+ > Add package from git URL...`
- Enter `https://github.com/Blockade-Games/BlockadeLabs-SDK-Unity.git`

## Pusher Package

The Blockade SDK can be used standalone or optionally together with a Pusher websockets package. If installed, the Pusher package will use websockets to listen for any changes in the Asset Generation Process. This changes the API request update strategy from Polling to Callbacks which will reduce the chances of being rate limited by the API.

You can learn more about the Pusher package here.

The Pusher library requires .NET Framework runtime:

- Set `Edit > Project Settings > Player > Api Compatibility Level` to `.NET Framework`.

Option 1: Install the Pusher package with git:

- Go to `Window > Package Manager`
- `+ > Add package from git URL...`
- Enter `https://github.com/pusher/pusher-websocket-unity.git#upm`

Option 2: Install the Pusher package with OpenUPM:

- CLI:
  - `openupm add com.pusher.pusherwebsocketunity`
- Registry:
  - Add scope: `com.pusher.pusherwebsocketunity`

> [!NOTE] After installing the Pusher package for Unity 2021+ versions you might encounter an error saying:

```
Assembly
'Packages/com.pusher.pusherwebsocketunity/Packages/PusherClient.2.1.0/lib/net472/P
usherClient.dll' will not be loaded due to errors: PusherClient references strong
named Newtonsoft.Json Assembly references: 12.0.0.0 Found in project: 13.0.0.0.
```

To resolve the issue go to `Edit > Project Settings > Player > Other Settings > Configuration > Assembly Version Validation` and disable `Version Validation`.

## Changelog

Refer to the changelog file here.

## Documentation

### Table of Contents

- Cancel Skybox Generation
- Cancel All Pending Skybox Generations

## Getting Started

After importing the package an onboarding window should pop up with directions to open the SkyboxAI scene.

If the onboarding window does not open for you, you can access the SkyboxAI scene by the following:

- menu: `Tools > Blockade Labs > Open SkyboxAI Scene`
- open: `Packages/Blockade Labs SDK/Scenes/SkyboxScene`

> [!NOTE] The SkyboxSceneAI scene uses Text Mesh Pro elements for runtime UI. If you haven't imported TMP Essentials you will be prompted to do so after you load the scene. When you are done importing TMP Essentials, reload the scene by either double clicking on it in the `Assets/Samples/Blockade Labs SDK/Scenes`.

The scene contains two notable gameObjects:

- `Blockade Labs Skybox Generator` generates skybox textures and materials.
- `Blockade Labs Skybox Mesh` generates and configures a mesh which combines the skybox with a generated depth map.

## Authentication

There are 4 ways to provide your API keys, in order of precedence:

> [!WARNING] We recommended using the environment variables to load the API key instead of having it hard coded in your source. It is not recommended use this method in production, but only for accepting user credentials, local testing and quick start scenarios.

1. Pass keys directly with constructor ⚠️
2. Unity Scriptable Object ⚠️
3. Load key from configuration file
4. Use System Environment Variables

**Pass keys directly with constructor**

```
var api = new BlockadeLabsClient("yourApiKey");
```

Or create a `BlockadeLabsAuthentication` object manually

```
var api = new BlockadeLabsClient(new BlockadeLabsAuthentication("yourApiKey"));
```

**Unity Scriptable Object**

You can save the key directly into a scriptable object that is located in the `Assets/Resources` folder.

You can create a new one by using the context menu of the project pane and creating a new `BlockadeLabsConfiguration` scriptable object.

`Create > BlockadeLabs > BlockadeLabsConfiguration`

#### Load key from configuration file

Attempts to load api keys from a configuration file, by default `.blockadelabs` in the current directory, optionally traversing up the directory tree or in the user's home directory.

To create a configuration file, create a new text file named `.blockadelabs` and containing the line:

##### Json format

```
{
  "apiKey": "yourApiKey",
}
```

You can also load the file directly with known path by calling a static method in Authentication:

```
var api = new BlockadeLabsClient(new
BlockadeLabsAuthentication().LoadFromDirectory("your/path/to/.blockadelabs"));;
```

#### Use System Environment Variables

Use your system's environment variables specify an api key to use.

- Use `BLOCKADELABS_API_KEY` for your api key.

```
var api = new BlockadeLabsClient(new
BlockadeLabsAuthentication().LoadFromEnvironment());
```

## Generating Skyboxes in Editor

You can generate a new skybox that will replace the existing one by following these steps.

1. Select the `Blockade Labs Skybox Generator` gameObject.
2. Add your Blockade Labs' `API key` in the designated inspector field.
3. Click the `Apply` button.
4. After the plugin is successfully initialized, some additional fields will become available.
5. Select the desired style.
6. Fill in the `Prompt` field.
7. Click the `Generate Skybox` Button.

8. Generated textures and materials are placed in the `Assets/Blockade Labs SDK` for use in your project.
9. You should see your new skybox in the game view. You can also click `Move Scene Camera to Skybox` to see the skybox in the scene view.

### Using the Skybox in your Scene

For detailed information on how skyboxes work in Unity, see using skyboxes.

#### Built-In Render Pipeline and Universal Render Pipeline (URP)

1. Go to `Window > Rendering > Lighting`.
2. Go to the `Environment` tab.
3. Drag the generated `skybox material` into the the `Skybox Material` field.
4. If you want to use the skybox as background lighting in your scene, ensure `Environment Lighting Source` is set to `Skybox`, then click `Generate Lighting`.

#### High-Definition Render Pipeline (HDRP)

1. Add a global volume to your scene: `GameObject > Volume > Global Volume`.
2. Drag the generated `HDRP volume profile` in to the `Profile` field.

### Mesh Creator

The `Blockade Labs Skybox Generator` component generates a color texture and a depth texture, which are assigned to the skybox material.

The `Blockade Labs Skybox Mesh` component generates a Tetrahedron mesh to apply this material. You can configure the `Mesh Density` and `Depth Scale` fields. The mesh and material will be configured to apply the generated depth map to deform the mesh.

Try zooming in and out with the scroll wheel in play mode to see the effect of the depth scale!

If you want to use the generated mesh in your own scene, click `Save Prefab`, then drag the new prefab into your scene.

## Generating Skyboxes at Runtime

If you're interested in having runtime generated content generated on demand, you can use the public API surface directly.

> [!WARNING] It is highly encouraged to use a proxy service when generating runtime content, to keep your API key secure and to minimize unauthorized content generation. **DO NOT** store or check in your API key into source control or ship it with your application. Ensure that you have not hard coded your API key in source control, and **DO NOT** check in your `BlockadeLabsConfiguration` into source control.

### BlockadeLabs API Proxy

BlockadeLabs-SDK-DotNet-Proxy v1.0.1

Using either the BlockadeLabs-SDK-DotNet or BlockadeLabs-SDK-Unity packages directly in your front-end app may expose your API keys and other sensitive information. To mitigate this risk, it is recommended to set up an intermediate API that makes requests to BlockadeLabs on behalf of your front-end app. This library can be utilized for both front-end and intermediary host configurations, ensuring secure communication with the BlockadeLabs API.

#### Front End Example

In the front end example, you will need to securely authenticate your users using your preferred OAuth provider. Once the user is authenticated, exchange your custom auth token with your API key on the backend.

Follow these steps:

1. Setup a new project using either the BlockadeLabs-SDK-DotNet or BlockadeLabs-SDK-Unity packages.
2. Authenticate users with your OAuth provider.
3. After successful authentication, create a new `BlockadeLabsAuthentication` object and pass in the custom token as your apiKey.
4. Create a new `BlockadeLabsClientSettings` object and specify the domain where your intermediate API is located.
5. Pass your new `auth` and `settings` objects to the `BlockadeLabsClient` constructor when you create the client instance.

Here's an example of how to set up the front end:

```
var authToken = await LoginAsync();
var auth = new BlockadeLabsAuthentication(authToken);
var settings = new BlockadeLabsClientSettings(domain: "api.your-custom-
domain.com");
using var api = new BlockadeLabsClient(auth, settings);
```

This setup allows your front end application to securely communicate with your backend that will be using the BlockadeLabs-SDK-DotNet-Proxy, which then forwards requests to the BlockadeLabs API. This ensures that your BlockadeLabs API keys and other sensitive information remain secure throughout the process.

#### Back End Example

In this example, we demonstrate how to set up and use `BlockadeLabsProxy` in a new ASP.NET Core web app. The proxy server will handle authentication and forward requests to the BlockadeLabs API, ensuring that your API keys and other sensitive information remain secure.

1. Create a new ASP.NET Core minimal web API project.
2. Add the BlockadeLabs-SDK-DotNet nuget package to your project.
    - Powershell install: `Install-Package BlockadeLabs-SDK-DotNet-Proxy`
    - Manually editing .csproj: `<PackageReference Include="BlockadeLabs-SDK-DotNet-Proxy" />`

3. Create a new class that inherits from `AbstractAuthenticationFilter` and override the `ValidateAuthentication` method. This will implement the `IAuthenticationFilter` that you will use to check user session token against your internal server.
4. In `Program.cs`, create a new proxy web application by calling `BlockadeLabsProxy.CreateWebApplication` method, passing your custom `AuthenticationFilter` as a type argument.
5. Create `BlockadeLabsAuthentication` as you would normally and load your API key from environment variable.

```
public partial class Program
{
    private class AuthenticationFilter : AbstractAuthenticationFilter
    {
        public override void ValidateAuthentication(IHeaderDictionary request)
        {
            // You will need to implement your own class to properly test
            // custom issued tokens you've setup for your end users.
            if (!request["x-api-key"].ToString().Contains(TestUserToken))
            {
                throw new AuthenticationException("User is not authorized");
            }
        }

        public override async Task ValidateAuthenticationAsync(IHeaderDictionary
request)
        {
            await Task.CompletedTask; // remote resource call

            // You will need to implement your own class to properly test
            // custom issued tokens you've setup for your end users.
            if (!request["x-api-key"].ToString().Contains(TestUserToken))
            {
                throw new AuthenticationException("User is not authorized");
            }
        }
    }

    public static void Main(string[] args)
    {
        var auth = BlockadeLabsAuthentication.LoadFromEnvironment();
        using var blockadeLabsClient = new BlockadeLabsClient(auth);
        BlockadeLabsProxy.CreateWebApplication<AuthenticationFilter>(args,
blockadeLabsClient).Run();
    }
}
```

Once you have set up your proxy server, your end users can now make authenticated requests to your proxy api instead of directly to the BlockadeLabs API. The proxy server will handle authentication and forward requests to the BlockadeLabs API, ensuring that your API keys and other sensitive information remain secure.

### Skyboxes API

#### Get Skybox Styles

Returns the list of predefined styles that can influence the overall aesthetic of your skybox generation.

```
var api = new BlockadeLabsClient();
var skyboxStyles = await
api.SkyboxEndpoint.GetSkyboxStylesAsync(SkyboxModel.Model3);

foreach (var skyboxStyle in skyboxStyles)
{
    Debug.Log($"{skyboxStyle.Name}");
}
```

#### Get Skybox Style Families

Returns the list of predefined styles that can influence the overall aesthetic of your skybox generation, sorted by style family. This route can be used in order to build a menu of styles sorted by family.

```
var api = new BlockadeLabsClient();
var skyboxFamilyStyles = await
BlockadeLabsClient.SkyboxEndpoint.GetSkyboxStyleFamiliesAsync(SkyboxModel.Model3);

foreach (var skyboxStyle in skyboxFamilyStyles)
{
    Debug.Log($"{skyboxStyle.Name}");
}
```

#### Get Skybox Export Options

Returns the list of all available export types.

```
var api = new BlockadeLabsClient();
var exportOptions = await api.SkyboxEndpoint.GetAllSkyboxExportOptionsAsync();

foreach (var exportOption in exportOptions)
{
    Debug.Log($"{exportOption.Id}: {exportOption.Name} | {exportOption.Key}");
}

var request = new SkyboxRequest("mars", enhancePrompt: true);
// Generates ALL export options for the skybox
var skyboxInfo = await api.SkyboxEndpoint.GenerateSkyboxAsync(request,
exportOptions);
Debug.Log($"Successfully created skybox: {skyboxInfo.Id}");
```

### Generate Skybox

Generate a skybox.

```
var api = new BlockadeLabsClient();
var skyboxStyles = await
BlockadeLabsClient.SkyboxEndpoint.GetSkyboxStylesAsync(SkyboxModel.Model3);
var request = new SkyboxRequest(skyboxStyles.First(), "mars", enhancePrompt:
true);

// You can also get progress callbacks when the generation progress has
changed/updated
var progress = new Progress<SkyboxInfo>(async progress =>
{
    Debug.Log(progress);
});

var skyboxInfo = await api.SkyboxEndpoint.GenerateSkyboxAsync(request,
progressCallback: progress);
Debug.Log($"Successfully created skybox: {skyboxInfo.Id}");

if (skyboxInfo.TryGetAsset<Texture2D>(SkyboxExportOption.Equirectangular_PNG, out
var texture))
{
    skyboxMaterial.mainTexture = texture;
}
```

### Get Skybox

Returns the skybox metadata for the given skybox id.

```
var skyboxId = 42;
var api = new BlockadeLabsClient();
var skyboxInfo = await api.SkyboxEndpoint.GetSkyboxInfoAsync(skyboxId);
Debug.Log($"Skybox: {result.Id}");
// Note: If you wish to use the returned skybox info textures you'll need to
additionally call await SkyboxInfo.LoadAssetsAsync(); before you can assign them
to a material property.
await skyboxInfo.LoadAssetsAsync();

if (skyboxInfo.TryGetAsset<Texture2D>(SkyboxExportOption.Equirectangular_PNG, out
var texture))
{
    skyboxMaterial.mainTexture = texture;
}
```

### Request Skybox Export

Exports the skybox with the requested export type.

> [!NOTE] You can also specify the export types when initially generating a skybox.

```
var skyboxId = 42;
var api = new BlockadeLabsClient();
var skyboxInfo = await api.SkyboxEndpoint.GetSkyboxInfoAsync(skyboxId);
skyboxInfo = await api.SkyboxEndpoint.ExportSkyboxAsync(skyboxInfo,
DefaultExportOptions.DepthMap_PNG);
await skyboxInfo.LoadAssetsAsync();

if (skyboxInfo.TryGetAsset<Texture2D>(SkyboxExportOption.DepthMap_PNG, out var
texture))
{
    skyboxMaterial.depthTexture = texture;
}
```

#### Delete Skybox

Deletes a skybox by id.

```
var skyboxId = 42;
var result = await api.SkyboxEndpoint.DeleteSkyboxAsync(skybox);
// result == true
```

#### Get Skybox History

Gets the previously generated skyboxes.

```
var history = await api.SkyboxEndpoint.GetSkyboxHistoryAsync();
Debug.Log($"Found {history.TotalCount} skyboxes");

foreach (var skybox in history.Skyboxes)
{
    Debug.Log($"{skybox.Id} {skybox.Title} status: {skybox.Status}");
}
```

#### Cancel Skybox Generation

Cancels a pending skybox generation request by id.

```
var skyboxId = 42;
var result = await CancelSkyboxGenerationAsync(skyboxId);
// result == true
```

> [!NOTE] This is automatically done when cancelling a skybox generation using cancellation token.

#### Cancel All Pending Skybox Generations

Cancels ALL pending skybox generation requests.

```
var result = await api.SkyboxEndpoint.CancelAllPendingSkyboxGenerationsAsync();
Debug.Log(result ? "All pending generations successfully cancelled" : "No pending
generations");
```