

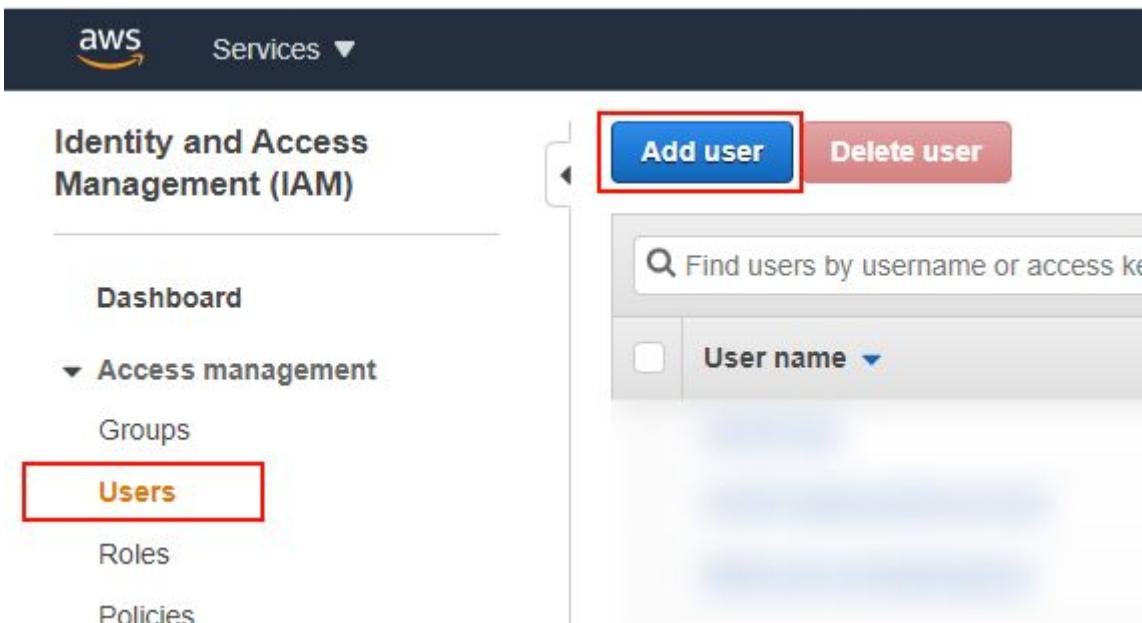
AWS SDK for Unity3d by Square Beam

Unity Friendly AWS SDK

Installation

Step 1. Create [AWS Profile](#) or [Login](#) in existing AWS Console

Step 2. Open [AWS AMI](#), hit “Users” in the left column and click “Add User”.



In the opened window set “User Name” and set “Access Type” to Programmatic Access.

Add user

1 2 3 4 5

Set user details

You can add multiple users at once with the same access type and permissions. [Learn more](#)

User name*

[+ Add another user](#)

Select AWS access type

Select how these users will access AWS. Access keys and autogenerated passwords are provided in the last step. [Learn more](#)

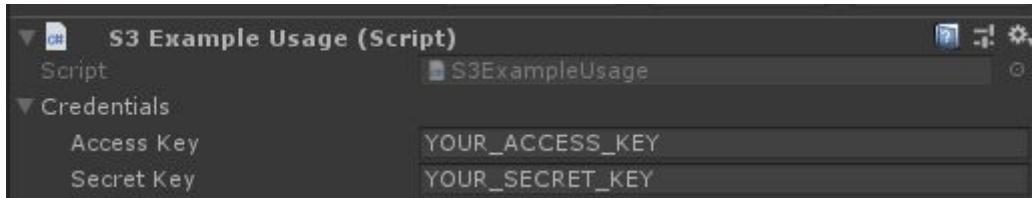
- Access type* **Programmatic access**
Enables an **access key ID** and **secret access key** for the AWS API, CLI, SDK, and other development tools.
- AWS Management Console access**
Enables a **password** that allows users to sign-in to the AWS Management Console.

A good practice is to have User-Per-Service

Download that **credentials.csv** file, Open it as a txt file and extract **aws_access_key** and **aws_secret_key** and store it in a secure place.

Do not include credentials in the build as a raw file! If it's not possible then enable IL2CPP Scripting backend and put in some class/struct. In this case, it will be harder to reverse-engineer.

Step 3. For example let's open **S3ExampleScene**, Select game object "Example" and in the Inspector window put extracted credentials in corresponding fields.



AWS S3

Step 1. Select your early created **IAM User** from the Installation paragraph and if you want to do a fast setup then please grant permission **S3FullAccess**.

Add user to group

The screenshot shows the 'Add user to group' step in the AWS IAM console. A search bar at the top right says 'Showing 13 results'. Below it is a table with columns 'Group' and 'Attached policies'. A row for 'S3' has a checkbox next to it. A row for 'S3FullAccess' has a checked checkbox and the policy name 'AmazonS3FullAccess' listed under 'Attached policies'. The 'Next: Review' button at the bottom right is highlighted with a red box.

Or you can add **Inline Policy**, in this case you can grant access only for specific buckets with specific permissions.

The screenshot shows the 'Summary' page for a user in the AWS IAM console. It includes fields for 'User ARN', 'Path' (set to '/'), and 'Creation time'. Below these are tabs for 'Permissions', 'Groups (2)', 'Tags', 'Security credentials', and 'Access Advisor'. The 'Permissions' tab is selected. Under 'Permissions policies', it says '(3 policies applied)'. There are two buttons at the bottom: 'Add permissions' and 'Add inline policy' (which is highlighted with a red box).

Step 2. Open **S3 Console** and hit **Create Bucket**. Provide **Bucket Name** and **Region**.

Create bucket

Buckets are containers for data stored in S3. [Learn more](#)

General configuration

Bucket name

Bucket name must be unique and must not contain spaces or uppercase letters. [See rules for bucket naming](#)

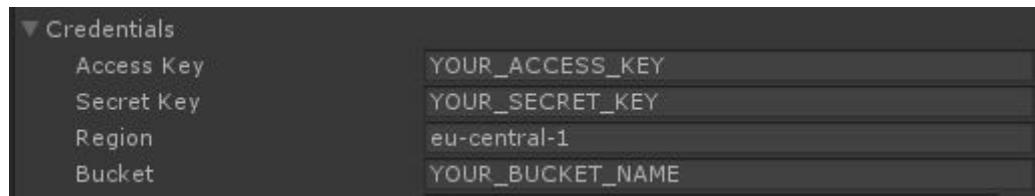
Region



Copy settings from existing bucket - optional

Only the bucket settings in the following configuration are copied.

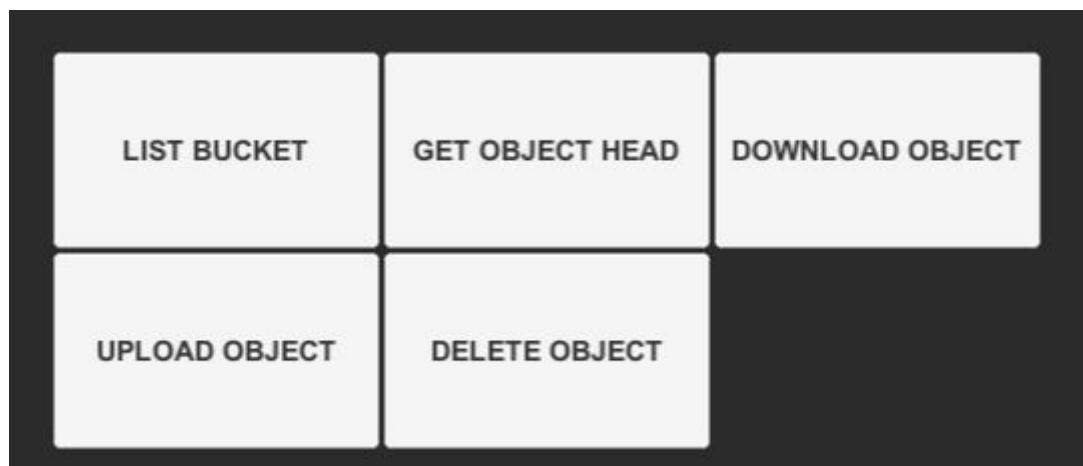
Region is very important. So for example if in UnityProject you set the wrong **Region** then it won't work! So for example if you created your bucket in the Frankfurt region then it will look like this in Inspector.



Step 3. Open S3ExampleScene, Select game object "Example" and in the Inspector window put Bucket Name and Region in corresponding fields.

Step 4. Upload test files in your early created bucket. You can find it under
[PathToProject|AWS|Services|S3|Example|TestFiles](#)

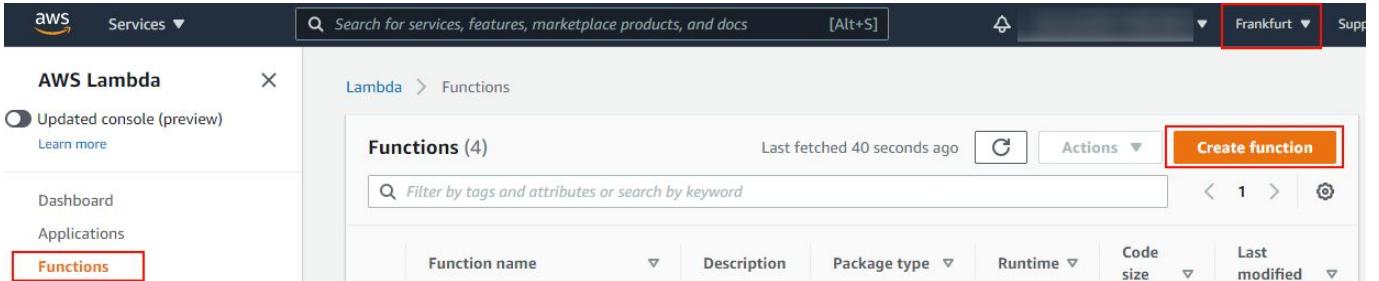
Step 5. Hit Play in Editor and Click on Buttons to work with it.



AWS Lambda

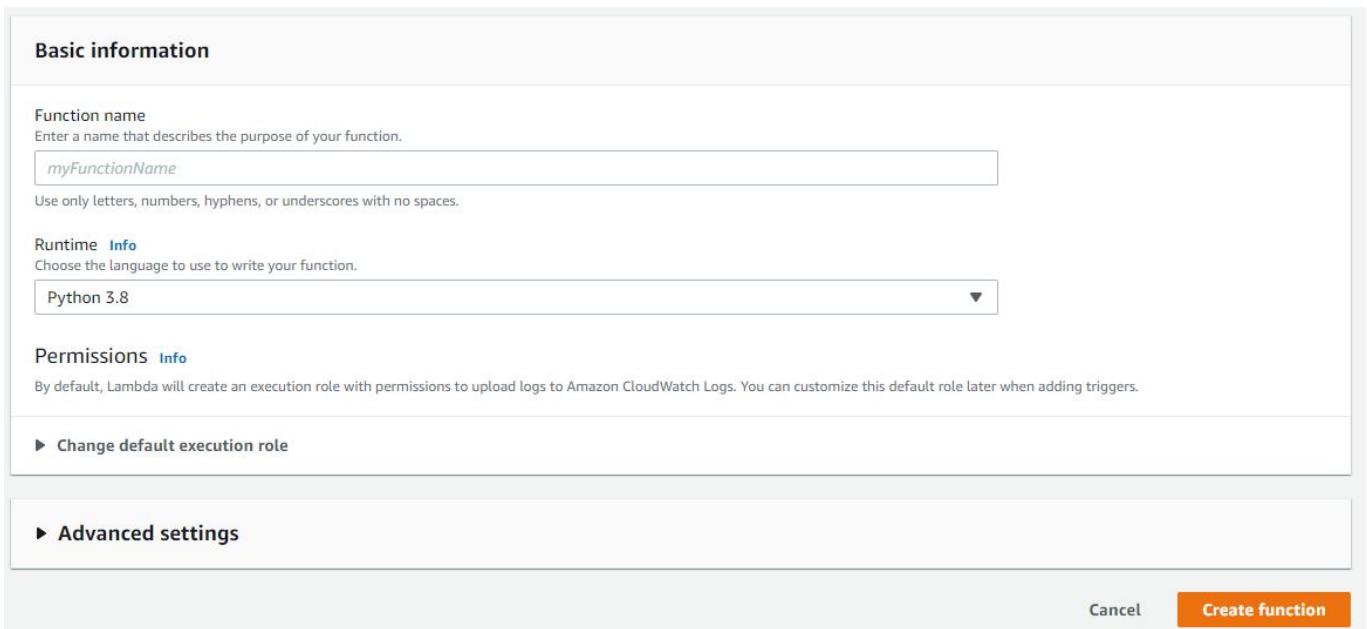
Step 1. Open [Lambda Console](#) and hit “Create function”.

Make sure that you are in the right Region (In my case it is Frankfurt eu-central-1)



The screenshot shows the AWS Lambda console interface. On the left, there's a sidebar with 'AWS Lambda' selected. The main area is titled 'Functions (4)'. At the top right, there's a 'Create function' button which is highlighted with a red box. The page also includes a search bar, a filter bar, and columns for 'Function name', 'Description', 'Package type', 'Runtime', 'Code size', and 'Last modified'.

Set **Function Name** and **Runtime** to **Python 3.8** everything else keep as it is and hit “Create function”

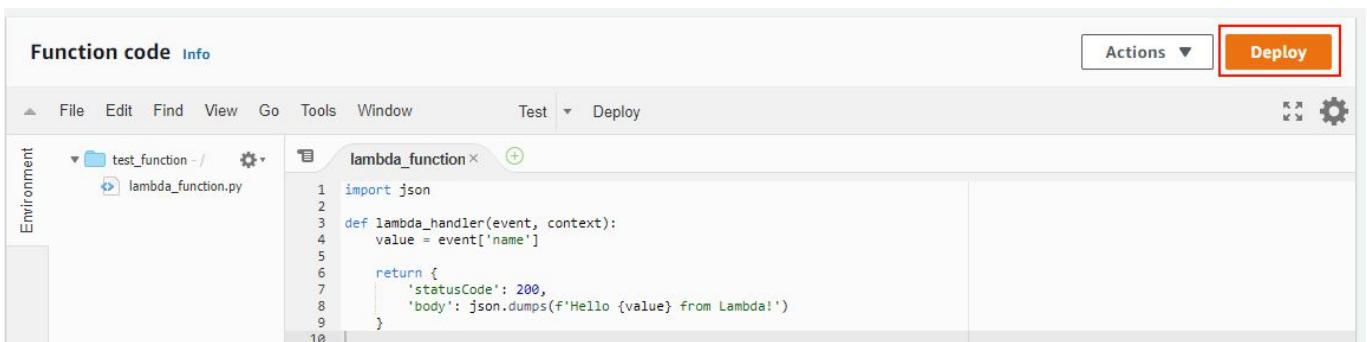


This screenshot shows the 'Basic information' step of the Lambda creation wizard. It includes fields for 'Function name' (set to 'myFunctionName') and 'Runtime' (set to 'Python 3.8'). There are also sections for 'Permissions' and 'Advanced settings'. At the bottom right, there are 'Cancel' and 'Create function' buttons, with 'Create function' highlighted by a red box.

Step 2. Replace default code on this simple “echo code” and hit **Deploy**

```
import json

def lambda_handler(event, context):
    value = event['name']
    return {
        'statusCode': 200,
        'body': json.dumps(f'Hello {value} from Lambda!')
    }
```



This screenshot shows the Lambda function editor. The 'Function code' tab is active. The code editor displays the Python code provided in the previous step. At the top right, there's a 'Deploy' button which is highlighted with a red box. The interface also includes a toolbar with 'File', 'Edit', 'Find', etc., and a sidebar for 'Environment'.

Step 3. Copy ARN of your function.

The screenshot shows the AWS Lambda Functions page. In the top navigation bar, 'Services' is selected. The search bar contains 'Search for services, features, marketplace products, and docs'. On the right, there are links for 'Frankfurt' and 'Support'. Below the navigation, the path 'Lambda > Functions > test_function' is shown. To the right of the path, the ARN of the function is displayed, with a red box highlighting the copy icon next to it.

Select your early created **IAM User** from the Installation paragraph and hit "**Add inline policy**".

The screenshot shows the IAM User permissions page. At the top, tabs for 'Permissions', 'Groups', 'Tags', 'Security credentials', and 'Access Advisor' are visible. Under the 'Permissions' tab, it says 'Permissions policies (2 policies applied)'. Below this are buttons for 'Add permissions' and 'Add inline policy', with the latter being highlighted by a red box.

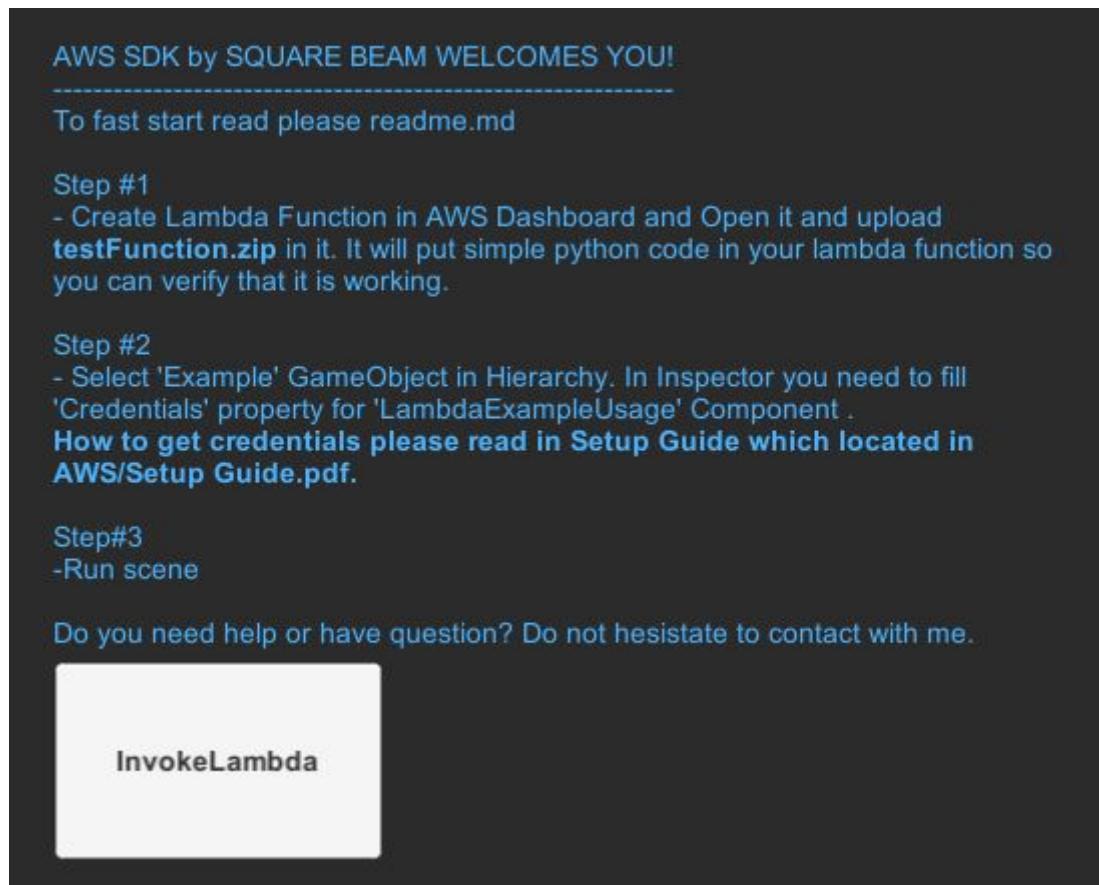
Step 4. We should grant InvokeLambda permission for this user. In the **Resources**, paste the ARN of your Function which you copied earlier and hit **Review policy**, give it **Name** and hit **Create Policy**.

The screenshot shows the 'Create policy' wizard, Step 1: Set permissions. It has two tabs: 'Visual editor' (selected) and 'JSON'. A link to 'Import managed policy' is also present. Below the tabs, there are 'Expand all' and 'Collapse all' buttons. The main area shows a policy structure with a single action: 'InvokeFunction' under the 'Lambda' service. The 'Actions' section is set to 'Write'. The 'Resources' section is set to 'Specific' and shows a field for 'function' with a placeholder 'function'. There is a 'Clone' and 'Remove' button at the top right of this section. Below the resources, there is a 'Request conditions' section with a note to 'Specify request conditions (optional)'. At the bottom right of the main area, there is a 'Clone' and 'Remove' button. A note at the bottom left states 'Character count: 527 of 2,048. The current character count includes character for all inline policies in the user: XelevateGraphWorker.' At the very bottom right are 'Cancel' and 'Review policy' buttons.

Step 5. Open **LambdaExampleScene**, select game object **Example**. In the Inspector window fill these credentials.

The screenshot shows the Unity Inspector window for a game object named 'Example'. Under the 'Script' section, there is a 'Credentials' group. It contains four fields: 'Access Key' with value 'YOUR_ACCESS_KEY', 'Secret Key' with value 'YOUR_SECRET_KEY', 'Region' with value 'YOUR_REGION', and 'Function' with value 'YOUR_LAMBDA_FUNCTION_NAME'. The 'YOUR...' values are placeholder text.

Step 6. Press Run and hit the InvokeLambda button.



AWS EC2

Create custom AMI

Step 1. Open **EC Console** Navigate to Instances and hit “**Launch Instances**”.

Make sure that you are in the right Region (In my case it is Frankfurt eu-central-1)

Choose AMI, If you are newbie choose Amazon Linux 2 AMI.

< < 1 to 32 of 32 AMIs > >

 Amazon Linux <small>Free tier eligible</small>	Amazon Linux 2 AMI (HVM), SSD Volume Type - ami-02cb52d7ba9887a93	Select
Amazon Linux 2 comes with five years support. It provides Linux kernel 4.14 tuned for optimal performance on Amazon EC2, systemd 219, GCC 7.3, Glibc 2.26, Binutils 2.29.1, and the latest software packages through extras. This AMI is the successor of the Amazon Linux AMI that is approaching end of life on December 31, 2020 and has been removed from this wizard.		
Root device type: ebs Virtualization type: hvm ENA Enabled: Yes		

Then select “**Instance Type**”, choose cheapest if you want just to test.

After you can hit **Review and Launch**, or you can specify more details if you want.

Navigate to **Instances**, select your runned instance and hit **Connect**, then you can install your software and establish processes. After you are finished with it **Stop** instance and hit **Actions->Image and Templates->Create Image**

The screenshot shows the AWS EC2 Instances page. At the top, there are buttons for 'Connect' and 'Launch instances'. A search bar labeled 'Filter instances' is present. Below the header, there are columns for 'Name', 'Instance ID', 'Instance state', and 'Instance type'. A single instance is listed, marked as 'Running' and of type 't3.nano'.

Operate Instances

Step 1. Open Unity Project, go to Scene “**EC2ExampleScene**” and Run It.

A Unity UI interface for launching an EC2 instance. It has three input fields: 'Enter Image-Id', 'Enter Count', and 'Enter Image Type'. Below these is a large button labeled 'Run EC2 Instance'.

In Image-Id set **AMI Id** which you created earlier, set **Count**, set **Instance Type**, (t3.micro) and hit **Run EC2 Instance**

Some Instance Types might not be available in specific regions.

Step 2. After success creation you will receive **InstanceId** which you can use in block below to operate Instances

A Unity UI interface for operating EC2 instances. It has a field labeled 'Enter Instance-Id' and four buttons: 'Describe EC2 Instance', 'Start EC2 Instance', 'Stop EC2 Instance', and 'Terminate EC2 Instance'.