Developer Guide

AWS Mobile SDK for Unity

AWS Mobile SDK for Unity: Developer Guide

Table of Contents

The AWS Mobile SDK for Unity is now included in the AWS SDK for .NET. This guide references the archived version of the Mobile SDK for Unity. For more information, see What is the AWS Mobile SDK for Unity?

What is the AWS Mobile SDK for Unity?

The AWS Mobile SDK for Unity is now included in the SDK for .NET. For more information, see the *AWS SDK for .NET Developer Guide*.

This guide is no longer updated—it references the archived version of the Mobile SDK for Unity.

Related guides and topics

- For front-end and mobile app development, we recommend using AWS Amplify.
- For special considerations for using the AWS SDK for .NET for your Unity apps, see Special considerations for Unity support in the *AWS SDK for .NET Developer Guide*.
- For reference purposes, you can find the archived version of the AWS Mobile SDK for Unity on GitHub.

Archived reference content

The archived Mobile SDK for Unity contains a set of .NET classes that enables games written with Unity to utilize AWS services. Applications written with the Mobile SDK for Unity can run on iOS or Android devices.

Supported AWS services include:

- Amazon Cognito
- Amazon DynamoDB
- AWS Identity and Access Management (IAM)
- Amazon Kinesis Data Streams
- AWS Lambda
- Amazon Mobile Analytics
- Amazon Simple Email Service (Amazon SES)
- Amazon Simple Notification Service (Amazon SNS)
- Amazon Simple Queue Service (Amazon SQS)
- Amazon Simple Storage Service (Amazon S3)

These services enable you to authenticate users, save player and game data, save objects in the cloud, send push notifications, and collect and analyze usage data.

Compatibility

The Mobile SDK for Unity v3 is compatible with Unity versions 4.6 and above.

The latest version of the Mobile SDK for Unity introduced improvements that might require you to change your code when you incorporate into your project. For more information about these changes, see Improvements in the AWS Mobile SDK for Unity on the Front-End Web & Mobile blog.

Download the Mobile SDK for Unity

You can also download the Mobile SDK for Unity as a .zip file here.

What's included in the Mobile SDK for Unity?

For the complete list of NuGet packages, samples, and other files in the Mobile SDK for Unity, see AWS SDK for .NET on GitHub.

Set Up the AWS Mobile SDK for Unity

To get started with the AWS Mobile SDK for Unity, you can set up the SDK and start building a new project, or you can integrate the SDK with an existing project. You can also clone and run the samples to get a sense of how the SDK works.

Prerequisites

Before you can use the AWS Mobile SDK for Unity, you will need the following:

- An AWS Account
- Unity version 4.x or 5.x (Unity 4.6.4p4 or Unity 5.0.1p3 is required if you want to write applications that run on iOS 64-bit)

After completing the prerequisites, you will need to do the following to get started:

1. Download the AWS Mobile SDK for Unity.
2. Configure the AWS Mobile SDK for Unity.
3. Obtain AWS credentials using Amazon Cognito.

Step 1: Download the AWS Mobile SDK for Unity

First, download the AWS Mobile SDK for Unity. Each package in the SDK is required to consume the corresponding AWS service based on the package's name. For example, the aws-unity-sdk-dynamodb-2.1.0.0.unitypackage package is used to call the AWS DynamoDB service. You can import all of the packages or just the ones you intend to use.

1. Open the Unity editor and create a new empty project, use the default settings.
2. Select **Assets** > **Import Package** > **Custom Package**.
3. In the **Import** package dialog, navigate to and select the .unitypackage files you want to use.
4. In the **Importing** package dialog, ensure all items are selected and click **Import**.

Step 2: Configure the AWS Mobile SDK for Unity

Create a Scene

When working with the AWS Mobile SDK for Unity, you can get started by including the following line of code in the `Start` or `Awake` method of your mono behavior class:

```
UnityInitializer.AttachToGameObject(this.gameObject);
```

Create your scene by choosing **New Scene** from the **File** menu.

The AWS SDK for Unity contains client classes for each AWS service it supports. These clients are configured using a file named **awsconfig.xml**. The following section describes the most commonly used settings in the **awsconfig.xml** file. For more information about these settings, see the Unity SDK API Reference.

Set the Default AWS Service Region

To configure the default region for all service clients:

```
<aws region="us-west-2" />
```

This sets the default region for all the service clients in the Unity SDK. This setting can be overridden by explicitly specifying the region at the time of creating an instance of the service client, like so:

```
IAmazonS3 s3Client = new AmazonS3Client(<credentials>,RegionEndpoint.USEast1);
```

Set Logging Information

Logging settings are specified as follows:

```
<logging logTo="UnityLogger"
         logResponses="Always"
         logMetrics="true"
         logMetricsFormat="JSON" />
```

This setting is used to configure logging in Unity. When you log to `UnityLogger`, the framework internally prints the output to the Debug Logs. If you want to log HTTP responses, set the

logResponses flag - the values can be Always, Never, or OnError. You can also log performance metrics for HTTP requests using the logMetrics property, the log format can be specified using LogMetricsFormat property, valid values are JSON or standard.

The following example shows the most commonly used settings in the awsconfig.xml file. For more information about specific service settings, see the service section below:

```
<?xml version="1.0" encoding="utf-8"?>
<aws region="us-west-2"
    <logging logTo="UnityLogger"
             logResponses="Always"
             logMetrics="true"
             logMetricsFormat="JSON" />
/>
```

Working with the link.xml file

The SDK uses reflection for platform-specific components. If you are using the IL2CPP scripting backend, `strip bytecode` is always enabled on iOS, so you need to have a `link.xml` file in your assembly root with the following entries:

```
<linker>
<!-- if you are using AWSConfigs.HttpClient.UnityWebRequest option-->
<assembly fullname="UnityEngine">
    <type fullname="UnityEngine.Networking.UnityWebRequest" preserve="all" />
    <type fullname="UnityEngine.Networking.UploadHandlerRaw" preserve="all" />
    <type fullname="UnityEngine.Networking.UploadHandler" preserve="all" />
    <type fullname="UnityEngine.Networking.DownloadHandler" preserve="all" />
    <type fullname="UnityEngine.Networking.DownloadHandlerBuffer" preserve="all" />
</assembly>
<assembly fullname="mscorlib">
    <namespace fullname="System.Security.Cryptography" preserve="all"/>
</assembly>
<assembly fullname="System">
    <namespace fullname="System.Security.Cryptography" preserve="all"/>
</assembly>
<assembly fullname="AWSSDK.Core" preserve="all"/>
<assembly fullname="AWSSDK.CognitoIdentity" preserve="all"/>
<assembly fullname="AWSSDK.SecurityToken" preserve="all"/>
add more services that you need here...
</linker>
```

Step 3: Obtain the Identity Pool ID using Amazon Cognito

To use AWS services in your mobile application, you must obtain the Identity Pool ID using Amazon Cognito Identity. Using Amazon Cognito to obtain the Identity Pool ID allows your app to access AWS services without having to embed your private credentials in your application. This also allows you to set permissions to control which AWS services your users have access to.

To get started with Amazon Cognito, you must create an identity pool. An identity pool is a store of user identity data specific to your account. Every identity pool has configurable IAM roles that allow you to specify which AWS services your application's users can access. Typically, a developer will use one identity pool per application. For more information on identity pools, see the Amazon Cognito Developer Guide.

To create an identity pool for your application:

1. Log in to the Amazon Cognito Console and click **Create new identity pool**.
2. Enter a name for your Identity Pool and check the checkbox to enable access to unauthenticated identities. Click **Create Pool** to create your identity pool.
3. Click **Allow** to create the two default roles associated with your identity pool–one for unauthenticated users and one for authenticated users. These default roles provide your identity pool access to Cognito Sync and Mobile Analytics.

The next page displays code that creates a credentials provider so you can easily integrate Cognito Identity with your Unity application. You pass the credentials provider object to the constructor of the AWS client you are using. The code looks like this:

```
CognitoAWSCredentials credentials = new CognitoAWSCredentials (
    "IDENTITY_POOL_ID", // Identity Pool ID
    RegionEndpoint.USEast1 // Region
);
```

Next Steps

- **Get Started**: Read Getting Started with the AWS Mobile SDK for Unity to get a more detailed overview of the services included in the SDK.

- **Run the demos**: View our sample Unity applications that demonstrate common use cases. To run the sample apps, set up the SDK for Unity as described above, and then follow the instructions contained in the README files of the individual samples.
- **Read the API Reference**: View the API Reference for the AWS Mobile SDK for Unity.
- **Ask questions**: Post questions on the AWS Mobile SDK Forums or open an issue on Github.

Getting Started with the AWS Mobile SDK for Unity

This page provides you with an overview of each AWS service in the AWS Mobile SDK for Unity, as well as instructions on how to set up Unity samples. You must complete all of the instructions on the Set Up the AWS Mobile SDK for Unity page before you start using the services below.

Amazon Cognito Identity

All calls made to AWS require AWS credentials. Rather than hard-coding your credentials into your apps, we recommend that you use Amazon Cognito Identity to provide AWS credentials to your application. Follow the instructions in Set Up the AWS Mobile SDK for Unity to obtain AWS credentials via Amazon Cognito.

Cognito also allows you to authenticate users using public log-in providers like Amazon, Facebook, Twitter, and Google as well as providers that support OpenID Connect. Cognito also works with unauthenticated users. Cognito provides temporary credentials with limited access rights that you specify with an Identity and Access Management (IAM) role. Cognito is configured by creating a new Identity Pool that is associated with an IAM role. The IAM role specifies the resources/services your app may access.

To get started with Cognito Identity, see the Amazon Cognito Developer Guide.

Amazon Cognito Sync

Cognito Sync makes it easy for you to save end users data such as user preferences or game state to the AWS Cloud so that it can be made available to users regardless of the device that they are using. Cognito can also save this data locally, allowing your apps to work even when an internet connection is not available. When an internet connection becomes available, your apps can sync their local data to the cloud.

To get started with Cognito Sync, see the Amazon Cognito Developer Guide.

Using the CognitoSyncManager sample

In the **Project** pane, navigate to **Assets/AWSSDK/examples/CognitoSync**, and in the right-hand side of the pane select the **CognitoSync** scene to open the scene.

To run the sample click the play button at the top of the editor screen. When the app runs it displays a few text boxes and buttons that allow you to enter some player information. Below this,

there are a series of buttons that save the player info locally, sync the local player info with the Cognito Cloud, refresh player info from the Cognito Cloud, and delete the local player info. Press each button to perform an operation. The sample displays feedback on the top of the game screen.

To configure the CognitoSyncManager sample, you must specify a Cognito Identity Pool ID. To specify this value, in the Unity editor, select **SyncManager** in the **Heirarchy** pane, and enter it into the **IDENTITY_POOL_ID** text box in the **Inspector Pane**.

> **Note**
>
> The CognitoSyncManager sample contains code that illustrates how use the Facebook identity Provider, search for the “USE_FACEBOOK_LOGIN” macro. This requires use of the Facebook SDK for Unity. For more information, see Facebook SDK for Unity.

Dynamo DB

Amazon DynamoDB is a fast, highly scalable, highly available, cost-effective, non-relational database service. DynamoDB removes traditional scalability limitations on data storage while maintaining low latency and predictable performance.

The AWS SDK for Unity provides both low-level and high-level libraries for working with DynamoDB. The high-level library includes the DynamoDB Object Mapper, which lets you map client-side classes to DynamoDB tables; perform various create, read, update, and delete (CRUD) operations; and execute queries. Using the DynamoDB Object Mapper, you can write simple, readable code that stores objects in the cloud.

For more information about DynamoDB, see DynamoDB Developer Guide.

For more information about using Dynamo DB from Unity Applications, see Amazon DynamoDB.

Using the DynamoDB Sample

In the **Project** pane, navigate to **Assets/AWSSDK/ examples/DynamoDB**. This sample is composed of the following scenes:

- DynamoDBExample - the initial scene of the app
- LowLevelDynamoDbExample - example using low-level DynamoDBD API
- TableQueryAndScanExample - example showing how to perform queries

- HighLevelExample - example using high-level DynamoDB API

Add these scenes into the build (in the order they appear above) by using the Build Settings dialog (open by selecting File.Build Settings). This sample create four tables: ProductCatalog, Forum, Thread, Reply.

To run the sample click the play button at the top of the editor screen. When the app runs it displays a number of buttons:

- Low Level Table Operations - illustrates how to create, list, update, describe, and delete tables.
- Mid Level Query & Scan Operations - illustrates how to perform queries.
- High Level Object Mapper - illustrates how to create, update, and delete objects.

Mobile Analytics

Using Amazon Mobile Analytics, you can track customer behaviors, aggregate metrics, generate data visualizations, and identify meaningful patterns. The AWS SDK for Unity provides integration with the Amazon Mobile Analytics service. For information about Mobile Analytics, see Mobile Analytics User Guide. For more information about using Mobile Analytics from Unity Applications, see Amazon Mobile Analytics.

Configuring Mobile Analytics

Mobile Analytics defines some settings that can be configured in the awsconfig.xml file:

```
<mobileAnalytics sessionTimeout = "5"
                 maxDBSize = "5242880"
                 dbWarningThreshold = "0.9"
                 maxRequestSize = "102400"
                 allowUseDataNetwork = "false"/>
```

- sessionTimeout - This the time interval after an application goes to background and when session can be terminated.
- maxDBSize - This is the size of the SQLIte Database. When the database reaches the maximum size, any additional events are dropped.
- dbWarningThreshold - This is the limit on the size of the database which, once reached, will generate warning logs.

- maxRequestSize - This is the maximum size of the request in Bytes that should be transmitted in an HTTP request to the mobile analytics service.
- allowUseDataNetwork - A boolean that specifies if the session events are sent on the data network.

Using the Mobile Analytics Sample

In the **Project** pane, navigate to **Assets/AWSSDK/ examples/Mobile Analytics**, and in the right-hand side of the pane select the **Amazon Mobile Analytics Sample** scene to open the scene. To use the sample, you need to add your app using the Amazon Mobile Analytics console. For more information about using the Mobile Analytics console, see Amazon Mobile Analytics User Guide.

Follow these steps to configure the sample before running:

1. Select the AmazonMobileAnalyticsSample game object.
2. Specify your App Id (created in the Amazon Mobile Analytics console) in the “App Id” field.
3. Specify your Cognito Identity Pool Id (created using the Amazon Cognito console at) in the “Cognito Identity Pool Id” field.
4. Ensure your authenticated and unauthenticated roles have permissions to access the Mobile Analytics service. For more information about applying policy to IAM Roles see, Managing Roles.

When running the sample application, be aware that events may not be transmitted to the backend service immediately. A background thread will buffer events locally and send them in batches to the Amazon Mobile Analytics backend at a regular interval (the default value is 60 seconds) to ensure your game’s performance is not adversely impacted. Due to the complex processing Amazon Mobile Analytics performs on your data, submitted events and corresponding reports may not be visible in the AWS console until up to 60 minutes after initial submission.

For more information on the reports provided by Amazon Mobile Analytics, see Report and Mobile Metrics.

Amazon S3

Amazon Simple Storage Service (Amazon S3), provides developers and IT teams with secure, durable, highly-scalable object storage. From Unity you can use S3 to store, list, and retrieve images, videos, music, and other data used by your games.

For more information about S3, see Amazon S3 and Getting Started with S3.

For more information about using S3 from Unity applications, see Amazon Simple Storage Service (S3).

Configuring the S3 Default Signature

The default S3 signature is configured as follows:

```
<s3 useSignatureVersion4="true" />
```

This is used to specify if you should use signature version 4 for S3 requests.

Using the S3 Sample

In the **Project** pane, navigate to **Assets/AWSSDK/examples/S3**, and in the right-hand side of the pane select the **S3Example** scene to open the scene. The sample illustrates how to list buckets, list objects within a bucket, post objects to a bucket and download objects from a bucket. Follow these steps to configure the sample before running:

1. Select the **S3** game object in the **Hierarchy** pane.
2. In the **Inspector** pane enter values for **S3BucketName** and **SampleFileName**. S3BucketName is the name of the bucket used by the sample and S3SampleFileName is the name of the file the sample will upload into the specified S3 bucket.
3. Ensure your authenticated and unauthenticated roles have permissions to access the S3 buckets in your account. For more information about applying policy to IAM Roles see, Managing Roles.

To run the sample click the play button at the top of the editor screen. When the app runs it displays a number of buttons:

- Get Objects - Gets a list of all objects in all buckets in your AWS account.
- Get Buckets - Gets a list of all buckets in your AWS account.
- Post Object - Uploads an object to a specified S3 bucket.
- Delete Object - Deletes all object from a specified S3 bucket.

The sample displays feedback on the top of the game screen.

Amazon Simple Notification Service

Amazon Simple Notification Service is a fast, flexible, fully managed push notification service that lets you send individual messages or to fan-out messages to large numbers of recipients. Amazon Simple Notification Service makes it simple and cost effective to send push notifications to mobile device users, email recipients or even send messages to other distributed services. To get started with Amazon Simple Notification Service, see Amazon Simple Notification Service.

AWS Lambda

AWS Lambda is a compute service that runs your code in response to requests or events and automatically manages the compute resources for you, making it easy to build applications that respond quickly to new information. AWS Lambda functions can be called directly from mobile, IoT, and Web apps and sends a response back synchronously, making it easy to create scalable, secure, and highly available backends for your mobile apps without the need to provision or manage infrastructure. For more information, see AWS Lambda.

Amazon Cognito Identity

What is Amazon Cognito Identity?

Using Amazon Cognito Identity, you can create unique identities for your users and authenticate them for secure access to your AWS resources like Amazon S3 or Amazon DynamoDB. Amazon Cognito Identity supports public identity providers—Amazon, Facebook, Twitter/Digits, Google, or any OpenID Connect-compatible provider—as well as unauthenticated identities. Cognito also supports developer authenticated identities, which let you register and authenticate users using your own backend authentication process, while still using Amazon Cognito Sync to synchronize user data and access AWS resources.

For more information on Cognito Identity, see the Amazon Cognito Developer Guide.

For information about Cognito Authentication Region availability, see Amazon Cognito Identity Region Availability.

Using a Public Provider to Authenticate Users

For information on using public identity providers like Amazon, Facebook, Twitter/Digits, or Google to authenticate users, see the External Providers in the Amazon Cognito Developer Guide.

Using Developer Authenticated Identities

For information on developer authenticated identities, see the Developer Authenticated Identities in the Amazon Cognito Developer Guide.

Amazon Cognito Sync

Cognito Sync is an AWS service and client library that enables cross-device syncing of application-related user data. You can use the Cognito Sync API to synchronize user data across devices. To use Cognito Sync in your app, you must include the AWS Mobile SDK for Unity in your project.

For instructions on how to integrate Amazon Coginto Sync in your application, see Amazon Cognito Sync Developer Guide.

Amazon Mobile Analytics

Using Amazon Mobile Analytics, you can track customer behaviors, aggregate metrics, generate data visualizations, and identify meaningful patterns. For information about Mobile Analytics see AWS Mobile Analytics.

Integrating Amazon Mobile Analytics

The sections below explain how to integrate Mobile Analytics with your app.

Create an App in the Mobile Analytics Console

Go to the Amazon Mobile Analytics console and create an app. Note the `appId` value, as you'll need it later.

> **Note**
>
> To learn more about working in the console, see the Amazon Mobile Analytics User Guide.

When you are creating an App in the Mobile Analytics Console you will need to specify a Cognito Identity Pool ID. To create a new Cognito Identity Pool and generate an ID, see Cognito Identity Developer Guide.

Integrate Mobile Analytics into Your App

To access Mobile Analytics from Unity you will need the following using statements:

```
using Amazon.MobileAnalytics.MobileAnalyticsManager;
using Amazon.CognitoIdentity;
```

Best practice is to use Amazon Cognito to provide temporary AWS credentials to your application. These credentials let the app access your AWS resources. To create a credentials provider, follow the instructions at Amazon Cognito Identity.

Instantiate a MobileAnalyticsManager instance with the following information:

- cognitoIdentityPoolId - The ID of the Cognito Identity Pool for your app

- cognitoRegion - The region for your Cognito Identity Pool, for example “RegionEndpoint.USEast1”
- region - The region for the Mobile Analytics service, for example “RegionEndpoint.USEast1”
- appId - The value generated by the Mobile Analytics Console when you add an app

Use the MobileAnalyticsClientContextConfig to initialize an **`MobileAnalyticsManager`** instance as shown in the following code snippet:

```
// Initialize the MobileAnalyticsManager
void Start()
{
    // ...
    analyticsManager = MobileAnalyticsManager.GetOrCreateInstance(
        new CognitoAWSCredentials(<cognitoIdentityPoolId>, <cognitoRegion>),
        <region>,
        <appId>);
    // ...
}
```

Note

The app ID is generated for you during the app creation wizard. Both of these values must match those in the Mobile Analytics Console.

The `appId` is used to group your data in the Mobile Analytics console. To find your app ID after creating the app in the Mobile Analytics console, browse to the Mobile Analytics Console, click the gear icon in the upper right-hand corner of the screen. This will display the App Management page which lists all registered apps and their app IDs.

Record Monetization Events

The SDK for Unity provides the `MonetizationEvent` class, which enables you generate monetization events to track purchases made within mobile applications. The following code snippet shows how to create a monetization event:

```
// Create the monetization event object
MonetizationEvent monetizationEvent = new MonetizationEvent();
```

```
// Set the details of the monetization event
monetizationEvent.Quantity = 3.0;
monetizationEvent.ItemPrice = 1.99;
monetizationEvent.ProductId = "ProductId123";
monetizationEvent.ItemPriceFormatted = "$1.99";
monetizationEvent.Store = "Your-App-Store";
monetizationEvent.TransactionId = "TransactionId123";
monetizationEvent.Currency = "USD";

// Record the monetiziation event
analyticsManager.RecordEvent(monetizationEvent);
```

Record Custom Events

Mobile Analytics allows you to define custom events. Custom events are defined entirely by you; they help you track user actions specific to your app or game. For more information about Custom events see Custom-Events. For this example, let's say your app is a game, and you want to record an event when a user completes a level. Create a "LevelComplete" event by creating a new `AmazonMobileAnalyticsEvent` instance:

```
CustomEvent customEvent = new CustomEvent("LevelComplete");

// Add attributes
customEvent.AddAttribute("LevelName","Level1");
customEvent.AddAttribute("CharacterClass","Warrior");
customEvent.AddAttribute("Successful","True");

// Add metrics
customEvent.AddMetric("Score",12345);
customEvent.AddMetric("TimeInLevel",64);

// Record the event
analyticsManager.RecordEvent(customEvent);
```

Recording Sessions

When the application loses focus you can pause the session. In `OnApplicationFocus` check to see if the app is being paused. If so call `PauseSession` otherwise call `ResumeSession` as shown in the following code snippet:

```
void OnApplicationFocus(bool focus)
```

```
{
    if(focus)
    {
        analyticsManager.ResumeSession();
    }
    else
    {
        analyticsManager.PauseSession();
    }
}
```

By default, if the user switches focus away from the app for less than 5 seconds, and switches back to the app the session will be resumed. If the user switches focus away from the app for 5 seconds or longer, a new session will be created. This setting is configurable in the awsconfig.xml file. For more information, refer to the Configuring Mobile Analytics section of Getting Started with the AWS Mobile SDK for Unity.

Amazon Simple Storage Service (S3)

Amazon Simple Storage Service (Amazon S3), provides developers and IT teams with secure, durable, highly-scalable object storage. Unity developers can take advantage of S3 to dynamically load assets used by their games. This can make games initially download quicker from app stores.

For more information about S3, see Amazon S3.

For information about AWS S3 Region availability, see AWS Service Region Availability.

> **Note**
>
> Some of the samples in this document assume the use of a text box variable called ResultText to display trace output.

Create and Configure an S3 Bucket

Amazon S3 stores your resources in Amazon S3 buckets - cloud storage containers that live in a specific region. Each Amazon S3 bucket must have a globally unique name. You can use the Amazon S3 Console to create a bucket.

Create an S3 Bucket

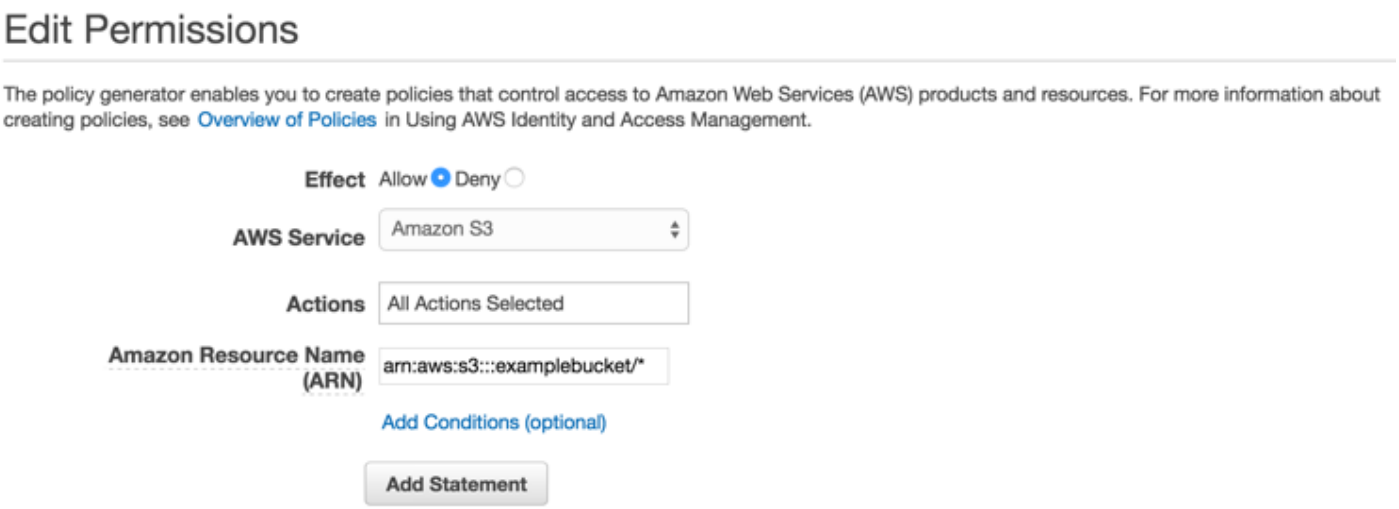
1. Sign in to the Amazon S3 console and click **Create Bucket**.
2. Enter a bucket name, select a region, and click **Create**.

Set Permissions for S3

The default IAM role policy grants your application access to Amazon Mobile Analytics and Amazon Cognito Sync. In order for your Cognito identity pool to access Amazon S3, you must modify the identity pool's roles.

1. Go to the Identity and Access Management Console and click **Roles** in the left-hand pane.
2. Type your identity pool name into the search box. Two roles will be listed: one for unauthenticated users and one for authenticated users.

3. Click the role for unauthenticated users (it will have unauth appended to your identity pool name).
4. Click **Create Role Policy**, select **Policy Generator**, and then click **Select**.
5. On the **Edit Permissions** page, enter the settings shown in the following image, replacing the Amazon Resource Name (ARN) with your own. The ARN of an S3 bucket looks like `arn:aws:s3:::examplebucket/*` and is composed of the region in which the bucket is located and the name of the bucket. The settings shown below will give your identity pool full to access to all actions for the specified bucket.



1. Click the **Add Statement** button and then click **Next Step**.
2. The Wizard will show you the configuration that you generated. Click **Apply Policy**.

For more information on granting access to S3, see Granting Access to an Amazon S3 Bucket.

Upload Files from the Console

To upload a test file to your bucket:

1. In the S3 console, in your bucket view, click **Upload**.
2. Click **Add Files** and select a test file to upload. For this tutorial, we'll assume you're uploading an image called `myImage.jpg`.
3. With your test image selected, click **Start Upload**.

(optional) Configure the Signature Version for S3 Requests

Every interaction with Amazon S3 is either authenticated or anonymous. AWS uses the Signature Version 4 or Signature Version 2 algorithms to authenticate calls to the service.

All new AWS regions created after January 2014 only support Signature Version 4. However, many older regions still support Signature Version 4 and Signature Version 2 requests.

If your bucket is in one of the regions that does not support Signature Version 2 requests as listed on this page, you must set the AWSConfigsS3.UseSignatureVersion4 property to “true”.

For more information on AWS Signature versions, see Authenticating Requests (AWS Signature Version 4).

Create the Amazon S3 Client

To use Amazon S3, we first need to create an AmazonS3Client instance which takes a reference to the CognitoAWSCredentials instance you created previously:

```
AmazonS3Client S3Client = new AmazonS3Client (credentials);
```

The `AmazonS3Client` class is the entry point to the high-level S3 API.

List Buckets

To list the buckets in an AWS account call the `AmazonS3Client.ListBucketsAsync` method as shown in the following sample code:

```
// ResultText is a label used for displaying status information
ResultText.text = "Fetching all the Buckets";
 Client.ListBucketsAsync(new ListBucketsRequest(), (responseObject) =>
 {
     ResultText.text += "\n";
     if (responseObject.Exception == null)
     {
         ResultText.text += "Got Response \nPrinting now \n";
         responseObject.Response.Buckets.ForEach((s3b) =>
         {
             ResultText.text += string.Format("bucket = {0}, created date = {1} \n",
             s3b.BucketName, s3b.CreationDate);
         });
```

```
    }
    else
    {
        ResultText.text += "Got Exception \n";
    }
});
```

List Objects

To list all of the objects in a bucket call the `AmazonS3Client.ListObjectsAsync` method as shown in the following sample code:

```
// ResultText is a label used for displaying status information
ResultText.text = "Fetching all the Objects from " + S3BucketName;

var request = new ListObjectsRequest()
{
    BucketName = S3BucketName
};

Client.ListObjectsAsync(request, (responseObject) =>
{
    ResultText.text += "\n";
    if (responseObject.Exception == null)
    {
        ResultText.text += "Got Response \nPrinting now \n";
        responseObject.Response.S3Objects.ForEach((o) =>
        {
            ResultText.text += string.Format("{0}\n", o.Key);
        });
    }
    else
    {
        ResultText.text += "Got Exception \n";
    }
});
```

Download an Object

To download an object, create a GetObjectRequest, specifying the bucket name and key and pass the object to a call to Client.GetObjectAsync:

```
private void GetObject()
{
    ResultText.text = string.Format("fetching {0} from bucket {1}",
    SampleFileName, S3BucketName);
    Client.GetObjectAsync(S3BucketName, SampleFileName, (responseObj) =>
    {
       string data = null;
       var response = responseObj.Response;
       if (response.ResponseStream != null)
       {
          using (StreamReader reader = new StreamReader(response.ResponseStream))
          {
             data = reader.ReadToEnd();
          }

          ResultText.text += "\n";
          ResultText.text += data;
       }
  });
}
```

GetObjectAsync takes an instance of the GetObjectRequest, a callback, and an AsyncOptions instance. The callback must be of type: `AmazonServiceCallback<GetObjectRequest, GetObjectResponse>`. The AsyncOptions instance is optional. If specified, it determines if the callback will run on the main thread.

Upload an Object

To upload an object, write your object to a stream, create a new PostObjectRequest and specify the key, bucket name and stream data.

The AWS SDK for Unity uses the WWW HTTP client which does not support the HTTP PUT operation. In order to upload an object to your S3 bucket, you need to use S3's Browser Post, as shown below.

```
public void PostObject(string fileName)
{
    ResultText.text = "Retrieving the file";

    var stream = new FileStream(Application.persistentDataPath +
    Path.DirectorySeparatorChar + fileName,
```

```
    FileMode.Open, FileAccess.Read, FileShare.Read);

    ResultText.text += "\nCreating request object";
    var request = new PostObjectRequest()
    {
        Bucket = S3BucketName,
        Key = fileName,
        InputStream = stream,
        CannedACL = S3CannedACL.Private
    };

    ResultText.text += "\nMaking HTTP post call";

    Client.PostObjectAsync(request, (responseObj) =>
    {
        if (responseObj.Exception == null)
        {
            ResultText.text += string.Format("\nobject {0} posted to bucket {1}",
            responseObj.Request.Key, responseObj.Request.Bucket);
        }
        else
        {
            ResultText.text += "\nException while posting the result object";
            ResultText.text += string.Format("\n receieved error {0}",
            responseObj.Response.HttpStatusCode.ToString());
        }
    });
}
```

Amazon DynamoDB

Amazon DynamoDB is a fast, highly scalable, highly available, cost-effective, non-relational database service. DynamoDB removes traditional scalability limitations on data storage while maintaining low latency and predictable performance. For information about DynamoDB, see Amazon DynamoDB.

The AWS Mobile SDK for Unity provides a high-level library for working with DynamoDB. You can also make requests directly against the low-level DynamoDB API, but for most use cases the high-level library is recommended. The AmazonDynamoDBClient is an especially useful part of the high-level library. Using this class, you can perform various create, read, update, and delete (CRUD) operations and execute queries.

> **Note**
>
> Some of the samples in this document assume the use of a text box variable called ResultText to display trace output.

Integrating Amazon DynamoDB

To use DynamoDB in a Unity application, you'll need to add the Unity SDK into your project. If you haven't already done so, download the SDK for Unity and follow the instructions in Set Up the AWS Mobile SDK for Unity. We recommend using Amazon Cognito Identity to provide temporary AWS credentials for your applications. These credentials allow your app to access AWS services and resources.

To use DynamoDB in an application, you must set the correct permissions. The following IAM policy allows the user to delete, get, put, scan, and update items in a specific DynamoDB table, which is identified by ARN:

```
{
"Statement": [{
    "Effect": "Allow",
    "Action": [
        "dynamodb:DeleteItem",
        "dynamodb:GetItem",
        "dynamodb:PutItem",
```

```
        "dynamodb:Scan",
        "dynamodb:UpdateItem"
    ],
    "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/MyTable"
}]
}
```

This policy should be applied to roles assigned to the Cognito identity pool, but you will need to replace the **`Resource`** valuewith the correct ARN for your DynamoDB table. Cognito automatically createsa role for your new identity pool, and you can apply policies to this role at the IAM console.

You should add or remove allowed actions based on the needs of your app. To learn more about IAM policies, see Using IAM. To learn more about DynamoDB-specific policies, see Using IAM to Control Access to DynamoDB Resources.

Create a DynamoDB Table

Now that we have our permissions and credentials set up, let's create a DynamoDB table for our application. To create a table, go to the DynamoDB console and follow these steps:

1. Click **Create Table**.
2. Enter `Bookstore` as the name of the table.
3. Select **Hash** as the primary key type.
4. Select **Number** and enter `id` for the hash attribute name. Click **Continue**.
5. Click **Continue** again to skip adding indexes.
6. Set the read capacity to `10` and the write capacity to `5`. Click **Continue**.
7. Enter a notification email and click **Continue** to create throughput alarms.
8. Click **Create**. DynamoDB will create your database.

Create a DynamoDB Client

For our app to interact with a DynamoDB table, we need a client. We can create a default DynamodDB client as follows:

```
var credentials = new CognitoAWSCredentials(IDENTITY_POOL_ID, RegionEndpoint.USEast1);
```

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient(credentials);
DynamoDBContext Context = new DynamoDBContext(client);
```

The AmazonDynamoDBClient class is the entry point for the DynamoDB API. The class provides instance methods for creating, describing, updating, and deleting tables, among other operations. Context adds a further layer of abstraction over the client and enables you to use additional functionality like the Object Persistance Model.

Describe a Table

To get a description of our DynamoDB table, we can use the following code:

```
resultText.text +=("\n*** Retrieving table information ***\n");
       var request = new DescribeTableRequest
       {
           TableName = @"ProductCatalog"
       };
       Client.DescribeTableAsync(request, (result) =>
       {
               if (result.Exception != null)
               {
                       resultText.text += result.Exception.Message;
                       Debug.Log(result.Exception);
                       return;
               }
               var response = result.Response;
               TableDescription description = response.Table;
               resultText.text += ("Name: " + description.TableName + "\n");
               resultText.text += ("# of items: " + description.ItemCount + "\n");
               resultText.text += ("Provision Throughput (reads/sec): " +
                   description.ProvisionedThroughput.ReadCapacityUnits + "\n");
               resultText.text += ("Provision Throughput (reads/sec): " +
                   description.ProvisionedThroughput.WriteCapacityUnits + "\n");

       }, null);
   }
```

In this example, we create a client and an DescribeTableRequest object, assign the name of our table to the **`TableName`** property, and then pass the request object to the DescribeTableAsync method on the AmazonDynamoDBClient object. DescribeTableAsync also takes a delegate that will be called when the async operation completes.

> **Note**
>
> All of the async methods on the AmazonDynamoDBClient take delegates that are called when the async operation completes.

Save an Object

To save an object to DynamoDB, use the SaveAsync<T> method of the AmazonDynamoDBClient object, where T is the type of object you are saving.

We called our database “Bookstore”, and in keeping with that theme we’ll implement a data model that records book-related attributes. Here are the classes that define our data model.

```
[DynamoDBTable("ProductCatalog")]
    public class Book
    {
        [DynamoDBHashKey]   // Hash key.
        public int Id { get; set; }
        [DynamoDBProperty]
        public string Title { get; set; }
        [DynamoDBProperty]
        public string ISBN { get; set; }
        [DynamoDBProperty("Authors")]    // Multi-valued (set type) attribute.
        public List<string> BookAuthors { get; set; }
    }
```

Of course, for a real bookstore application we’d need additional fields for things like author and price. The Book class is decorated with the [DynamoDBTable] attribute, this defines the database table objects of type Book will be written to. The key for each instance of the Book class is identified using the [DynamoDBHashKey] attribute. Properties are identified with the [DynamoDBProperty] attribute, these specify the column in the database table to which the property will be written. With the model in place, we can write some methods to create, retrieve, update, and delete Book objects.

Create a Book

```
private void PerformCreateOperation()
{
```

```
    Book myBook = new Book
    {
        Id = bookID,
        Title = "object persistence-AWS SDK for.NET SDK-Book 1001",
        ISBN = "111-1111111001",
        BookAuthors = new List<string> { "Author 1", "Author 2" },
    };

    // Save the book.
    Context.SaveAsync(myBook,(result)=>{
        if(result.Exception == null)
            resultText.text += @"book saved";
    });
}
```

Retrieve a Book

```
private void RetrieveBook()
{
    this.displayMessage += "\n*** Load book**\n";
    Context.LoadAsync<Book>(bookID,
                            (AmazonDynamoResult<Book> result) =>
    {
        if (result.Exception != null)
        {

            this.displayMessage += ("LoadAsync error" +result.Exception.Message);
            Debug.LogException(result.Exception);
            return;
        }
        _retrievedBook = result.Response;
        this.displayMessage += ("Retrieved Book: " +
                                "\nId=" + _retrievedBook.Id +
                                "\nTitle=" + _retrievedBook.Title +
                                "\nISBN=" + _retrievedBook.ISBN);
        string authors = "";
        foreach(string author in _retrievedBook.BookAuthors)
            authors += author + ",";
        this.displayMessage += "\nBookAuthor= "+ authors;
        this.displayMessage += ("\nDimensions= "+ _retrievedBook.Dimensions.Length + "
 X " +
                                _retrievedBook.Dimensions.Height + " X " +
```

```
                                    _retrievedBook.Dimensions.Thickness);

    }, null);
}
```

Update a Book

```
private void PerformUpdateOperation()
{
    // Retrieve the book.
    Book bookRetrieved = null;
    Context.LoadAsync<Book>(bookID,(result)=>
    {
        if(result.Exception == null )
        {
            bookRetrieved = result.Result as Book;
            // Update few properties.
            bookRetrieved.ISBN = "222-2222221001";
            // Replace existing authors list with this
            bookRetrieved.BookAuthors = new List<string> { "Author 1", "Author x" };
            Context.SaveAsync<Book>(bookRetrieved,(res)=>
            {
                if(res.Exception == null)
                     resultText.text += ("\nBook updated");
            });
        }
   });
}
```

Delete a Book

```
private void PerformDeleteOperation()
{
    // Delete the book.
    Context.DeleteAsync<Book>(bookID,(res)=>
    {
        if(res.Exception == null)
        {
            Context.LoadAsync<Book>(bookID,(result)=>
            {
                 Book deletedBook = result.Result;
```

```
                if(deletedBook==null)
                    resultText.text += ("\nBook is deleted");
            });
        }
    });
}
```

Amazon Simple Notification Service

Using Amazon Simple Notification Service (SNS) and the Unity SDK, you can write iOS and Android apps that can receive mobile push notifications. For information about SNS, see Amazon Simple Notification Service.

This topic will walk you through configuring the AWS SDK for Unity sample app, SNSExample.unity, to receive mobile push notifications through Amazon SNS.

You can create both iOS and Android apps using the SNSExample.unity sample. The configuration steps are different between iOS and Android please read the appropriate section below for the platform you are targeting.

Prerequisites

The following prerequisites are required for using this solution.

Set Permissions for SNS

When you create a Cognito Identity Pool two IAM roles are generated:

- Cognito/_<Identity-Pool-Name>Auth_DefaultRole - the default IAM role for authenticated users
- Cognito/_<Identity-Pool-Name>Unauth_DefaultRole - the default IAM role for unauthenticated users

You must add permissions to access the Amazon SNS service to these roles. To do this:

1. Browse to the IAM Console and select the IAM role to configure.
2. Click **Attach Policy**, select the AmazonSNSFullAccess policy and click **Attach Policy**.

> **Note**
>
> Using AmazonSNSFullAccess is not recommended in a production environment, we use it here to allow you to get up and running quickly. For more information about specifying permissions for an IAM role, see Overview of IAM Role Permissions.

iOS Prerequisites

- Membership in the Apple iOS Developer Program
- Generate a signing identity
- Create a provisioning profile configured for push notifications

You will need to run your app on a physical device to receive push notifications. To run your app on a device you must have a membership in the Apple iOS Developer Program Membership. Once you have a membership, you can use Xcode to generate a signing identity. For more information, see Apple's App Distribution Quick Start documentation. Next you will need a provisioning profile configured for push notifications for more information, see Apple's Configuring Push Notifications documentation.

Android Prerequisites

- Install the Android SDK
- Install the JDK
- android-support-v4.jar
- google-play-services.jar

Configuring the Unity Sample App for iOS

Open the Unity editor and create a new project. Import the AWS SDK for Unity package by selecting **Assets/Import Package/Custom Package** and selecting aws-unity-sdk-sns-2.0.0.1.unitypackage. Ensure all items in the **Importing Package** dialog are selected and click **Import**.

Unity Configuration

Perform the following steps to configure the Unity project:

1. In the **Project** pane, navigate to **Assets/AWSSDK/examples** and open the SNSExample scene.
2. In the **Hierarchy** pane, select SNSExample.
3. In the **Inspector** pane specify your Cognito Identity Pool ID.
4. Notice there is a text box for **iOS Platform Application ARN**, you will generate that information later on.

5. Select **File/Build Settings**, in the **Build Settings** dialog, click the **Add Current** button below the **Scenes in Build** list box to add the current scene to the build.
6. Under **Platform** select **iOS** and click the **Player Settings...** button, in the **Inspector Pane** of the Unity editor, click the iPhone icon and scroll down to the **Identification** section and specify a **Bundle Identifier**.

iOS Configuration

Perform the following steps to configure the sample to configure iOS specific settings:

1. In a web browser, go to the Apple Developer Member Center, click **Certificates, Identifiers & Profiles**.
2. Click **Identifiers** under **iOS Apps**, click the plus button in the upper right-hand corner of the web page to add a new iOS App ID, and enter an App ID description.
3. Scroll down to the **Add ID Suffix** section and select **Explicit App ID** and enter your bundle identifier.
4. Scroll down to the **App Services** section and select **Push Notifications**.
5. Click the **Continue** button.
6. Click the **Submit** button.
7. Click the **Done** button.
8. Select the App ID you just created and then click the **Edit** button.
9. Scroll down to the **Push Notifications** section.
10. Click the **Create Certificate** button under **Development SSL Certificate**.
11. Follow the instructions to create a Certificate Signing Request (CSR), upload the request, and download an SSL certificate that will be used to communicate with Apple Notification Service (APNS).
12. Back in the **Certificates, Identifiers & Profiles** web page, click **All** under **Provisioning Profiles**.
13. Click the plus button in the upper right-hand corner to add a new provisioning profile.
14. Select **iOS App Development** and click the **Continue** button.
15. Select your App ID and click the **Continue** button.
16. Select your developer certificate and click the **Continue** button.
17. Select your device and click the **Continue** button.
18. Enter a profile name and click the **Generate** button.

19. Download and double click the provision file to install the provisioning profile.

You may need to refresh the Provisioning Profiles in Xcode after adding a new one. In Xcode:

1. Select the **Xcode/Preferences** menu item.
2. Select the **Accounts** tab, select your Apple ID and click **View Details**.
3. Click the refresh button in the lower left-hand corner of the dialog to refresh your provisioning profiles and make sure your new profile is displayed.

SNS Configuration

1. Run the KeyChain access app, select **My Certificates** on the lower left-hand side of the screen, right click the SSL certificate you generated to connect to APNS and select **Export**, you will be prompted to specify a name for the file and a password to protect the certificate. The certificate will be saved in a P12 file.
2. In a web browser go to the SNS Console and click **Applications** on the left-hand side of the screen.
3. Click **Create platform application** to create a new SNS platform application.
4. Enter an **Application Name**.
5. Select **Apple Push Notification Service Sandbox (APNS_SANDBOX)** for **Push notification platform**.
6. Click **Choose File** and select the P12 file you created when you exported your SSL certificate.
7. Enter the password you specified when you exported the SSL certificate and click **Load Credentials From File**.
8. Click **Create platform application**.
9. Select the Platform Application you just created and copy the Application ARN.
10. Go back to your project in the Unity Editor, select **SNSExample** in the **Hierarchy** pane, in the **Inspector** pane and paste the Platform Application ARN into the text box labeled **iOS Platform Application ARN**.
11. Select **File/Build Settings** and click the **Build** button this will create an Xcode project.

Using Xcode

1. Open the Xcode project, and select the project in the Project Navigator.
2. Verify the bundle identifier is set correctly
3. Verify your Apple Developer Account is specified in the **Team** - this is required for your Provisioning Profile to take effect.
4. Build the project and run it on your device.
5. Tap the **Register for Notification**, tap **OK** to allow notifications, the app will display your device token

In the SNS Console, click **Applications**, select your platform application and click **Create Platform Endpoint**, and enter the device token displayed by the app.

At this point your app, APNS, and NSN are fully configured. You can select your platform application, select your endpoint, and click **Publish to endpoint** to send a push notification to your device.

Unity Sample (iOS)

The sample creates an CognitoAWSCredentials instance to generate temporary limited-scope credentials that allows the app to call AWS services. It also creates an instance of AmazonSimpleNotificationServiceClient to communicate with SNS. The app displays two buttons labeled **Register for Notification** and **Unregister**.

When the **Register for Notifications** button is tapped, the `RegisterDevice()` method is called. `RegisterDevice()` calls `UnityEngine.iOS.NotificationServices.RegisterForNotifications`, which specifies which notification types (alert, sound, or badge) will be used. It also makes an asynchronous call to APNS to get a device token. Because there is no callback defined, `CheckForDeviceToken` is called repeatedly (up to 10 times) to check for the device token.

When a token is retrieved `AmazonSimpleNotificationServiceClient.CreatePlatformEndpointAsync()` is called to create an endpoint for the SNS platform application.

The sample is now configured to receive push notifications. You can browse to the SNS Console, click **Applications** on the left-hand side of the page, select your platform application, select an

endpoint, and click **Publish to endpoint**. Select the endpoint to use and click **Publish to Endpoint**. Type in a text message in the text box and click **Publish message** to publish a message.

Configuring the Unity Sample App for Android

Open the Unity editor and create a new project. Import the AWS SDK for Unity package by selecting **Assets/Import Package/Custom Package** and selecting aws-unity-sdk-sns-2.0.0.1.unitypackage. Ensure all items in the **Importing Package** dialog are selected and click **Import**.

Unity Configuration

Perform the following steps to configure the Unity project:

1. In the **Project** pane, navigate to **Assets/AWSSDK/examples** and open the SNSExample scene.
2. In the **Hierarchy** pane, select SNSExample.
3. In the **Inspector** pane specify your Cognito Identity Pool ID.
4. Notice there is a text box for **Android Platform Application ARN** and **Google Console Project ID**, you will generate that information later on.
5. Select **File/Build Settings**, in the **Build Settings** dialog, click the **Add Current** button below the **Scenes in Build** list box to add the current scene to the build.
6. Under **Platform** select **Android** and click the **Player Settings...** button, in the **Inspector Pane** of the Unity editor, click the Android icon and scroll down to the **Identification** section and specify a **Bundle Identifier**.
7. Copy android-support-v4.jar and google-play-services.jar into the **Assets/Plugins/Android** directory in the **Project** pane.

For more information about where to find android-support-v4.jar, see Android Support Library Setup. For more information about how to find google-play-services.jar, see Google APIs for Android Setup.

Android Configuration

First add a new Google API project:

1. In a web browser, go to the Google Developers Console, click **Create Project**.

2. In the **New Project** box, enter a project name, take note of the project number (you will need it later) and click **Create**.

Next, enable the Google Cloud Messaging (GCM) service for your project:

1. In the Google Developers Console, your new project should already be selected, if not, select it in the drop down at the top of the page.
2. Select **APIs & auth** from the side bar on the left-hand side of the page.
3. In the search box, type “Google Cloud Messaging for Android” and click the **Google Cloud Messaging for Android** link below.
4. Click **Enable API**.

Finally obtain an API Key:

1. In the Google Developers Console, select **APIs & auth** > **Credentials**.
2. Under **Public API access**, click **Create new key**.
3. In the **Create a new key** dialog, click **Server key**.
4. In the resulting dialog, click **Create** and copy the API key displayed.

You will use the API key to perform authentication later on.

SNS Configuration

1. In a web browser go to the SNS Console and click **Applications** on the left-hand side of the screen.
2. Click **Create platform application** to create a new SNS platform application.
3. Enter an **Application Name**
4. Select **Google Cloud Messaging (GCM)** for **Push notification platform**
5. Paste the API key into the text box labeled **API key**.
6. Click **Create platform application**
7. Select the Platform Application you just created and copy the Application ARN.
8. Go back to your project in the Unity Editor, select **SNSExample** in the **Hierarchy** pane, in the **Inspector** pane and paste the Platform Application ARN into the text box labeled **Android**

Platform Application ARN and your project number into the text box labeled **Google Console Project ID**.

9. Connect your Android device to your computer, select **File/Build Settings** and click the **Build and Run.**

Unity Sample (Android)

The sample creates an CognitoAWSCredentials instance to generate temporary limited-scope credentials that allows the app to call AWS services. It also creates an instance of AmazonSimpleNotificationServiceClient to communicate with SNS.

The app displays two buttons labeled **Register for Notification** and **Unregister**. When the **Register for Notifications** button is tapped, the `RegisterDevice()` method is called. `RegisterDevice()` calls `GCM.Register`, which registers the app with GCM. GCM is a class defined within the example code. It makes an asynchronous call to register the app with GCM.

When the callback is called `AmazonSimpleNotificationServiceClient.CreatePlatformEndpointAsync` is called to create a platform endpoint to receive SNS messages.

The sample is now configured to receive push notifications. You can browse to the SNS Console, click **Applications** on the left-hand side of the page, select your platform application, select an endpoint, and click **Publish to endpoint**. Select the endpoint to use and click **Publish to Endpoint**. Type in a text message in the text box and click **Publish message** to publish a message.

AWS Lambda

AWS Lambda is a compute service that runs your code in response to requests or events and automatically manages the compute resources for you, making it easy to build applications that respond quickly to new information. AWS Lambda functions can be called directly from mobile, IoT, and Web apps and sends a response back synchronously, making it easy to create scalable, secure, and highly available backends for your mobile apps without the need to provision or manage infrastructure.

AWS Lambda can execute your Lambda functions in response to one of the following:

- Events, such as discrete updates (for example, object-created events in Amazon S3 or CloudWatch alerts), or streaming updates (for example, website clickstreams or outputs from connected devices).
- JSON inputs or HTTPS commands from your custom applications.

AWS Lambda executes your code only when needed and scales automatically, from a few requests per day to thousands per second. With these capabilities, you can use Lambda to easily build triggers for AWS services like Amazon S3 and Amazon DynamoDB, process streaming data stored in Amazon Kinesis, or create your own back-end that operates at AWS scale, performance, and security.

To learn more about how AWS Lambda works, see AWS Lambda: How It Works.

Permissions

There are two types of permissions related to Lambda functions:

- **Execution permissions** — The permissions that your Lambda function needs to access other AWS resources in your account. You grant these permissions by creating an IAM role, known as an execution role.
- **Invocation permissions** — The permissions that the event source needs to communicate with your Lambda function. Depending on the invocation model (push or pull model), you can grant these permissions using either the execution role or resource policies (the access policy associated with your Lambda function).

Project Setup

Set Permissions for AWS Lambda

1. Open the AWS IAM Console.
2. Attach this custom policy to your roles, which allows your application to make calls to AWS Lambda.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "lambda:*"
      ],
      "Resource": "*"
    }
  ]
}
```

Create a New Execution Role

This role applies to the Lambda function that you will create in the next step and determines which AWS resources that function can access.

1. Open the AWS IAM Console.
2. Click **Roles**.
3. Click **Create New Roles**.
4. Follow the on-screen instructions to select the services and corresponding policies that your Lambda function will need access to. For example, if you want your Lambda function to create an S3 bucket, your policy will need write access to S3.
5. Click **Create Role**.

Creating a Function in AWS Lambda

1. Open the AWS Lambda Console.
2. Click **Create a Lambda function**.
3. Click **Skip** to skip creating a blueprint.
4. Configure your own function on the next screen. Enter the function name, a description, and choose your runtime. Follow the on-screen instructions based on your chosen runtime. Specify your execution permissions by assigning your newly created execution role to your function.
5. When finished, click **Next**.
6. Click **Create Function**.

Create a Lambda Client

```
var credentials = new CognitoAWSCredentials(IDENTITY_POOL_ID, RegionEndpoint.USEast1);
var Client = new AmazonLambdaClient(credentials, RegionEndpoint.USEast1);
```

Create a Request Object

Create a request object to specify the invocation type and the function name:

```
var request = new InvokeRequest()
{
    FunctionName = "hello-world",
    Payload = "{\"key1\" : \"Hello World!\"}",
    InvocationType = InvocationType.RequestResponse
};
```

Invoke Your Lambda Function

Call invoke, passing the request object:

```
Client.InvokeAsync(request, (result) =>
{
    if (result.Exception == null)
    {
        Debug.Log(Encoding.ASCII.GetString(result.Response.Payload.ToArray()));
```

```
    }
    else
    {
        Debug.LogError(result.Exception);
    }
});
```

Troubleshooting

Due to limitations of the Unity.WWW class used by the AWS SDK for Unity, detailed error messages are not returned when a problem occurs while calling an AWS service. This topic describes some ideas for troubleshooting such problems.

Ensure IAM Role Has Required Permissions

When calling AWS services your app uses an identity from a Cognito identity pool. Each identity in the pool is associated with an IAM (Identity and Access Management) role. The role has one or more policy files associated with it that specify what AWS resources the users assigned to the role have access to. By default two roles are created, one for authenticated users, and one for unauthenticated users. You will need to either modify the existing policy file or associate a new policy file with the permisssions required by your app. If your app allows both authenticated and unauthenticated users, both roles must be granted permissions for accessing the AWS resources your app needs.

The following policy file shows how to grant access to an S3 bucket:

```
{
  "Statement": [
    {
      "Action": [
        "s3:AbortMultipartUpload",
        "s3:DeleteObject",
        "s3:GetObject",
        "s3:PutObject"
      ],
      "Effect": "Allow",
      "Resource": "arn:aws:s3:::MYBUCKETNAME/*",
      "Principal": "*"
    }
  ]
}
```

The following policy file shows how to grant access to a DynamoDB database:

```
{
"Statement": [{
```

```
    "Effect": "Allow",
    "Action": [
        "dynamodb:DeleteItem",
        "dynamodb:GetItem",
        "dynamodb:PutItem",
        "dynamodb:Scan",
        "dynamodb:UpdateItem"
    ],
    "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/MyTable"
}]
}
```

For more information about specifying policies, see IAM Policies.

Using a HTTP Proxy Debugger

If the AWS service your app is calling has an HTTP or HTTPS endpoint, you can use an HTTP/HTTPS proxy debugger to view the requests and responsese to gain more insight into what is occuring. There are a number of HTTP proxy debuggers available such as:

- Charles - a web debugging proxy for OSX
- Fiddler - a web debugging proxyfidd for Windows

> **Important**
>
> There is a known issue with the Cognito Credential Provider when running the Charles web debugging proxy that prevents the credential provider from working correctly.

Both Charles and Fiddler require some configuration to be able to view SSL encrypted traffic, please read the documentation for these tools for further information. If you are using a web debugging proxy that cannot be configured to display encrypted traffic, open the aws_endpoints_json file (located in AWSUnitySDK/AWSCore/Resources) and set the HTTP tag for the AWS service you need to debug to true