

---

# AWS Mobile Hub

## Developer Guide

### Version 1.0





## **AWS Mobile Hub: Developer Guide**

Copyright © 2016 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

## Table of Contents

What Is AWS Mobile Hub? .....	1
How Can I Use AWS Mobile Hub? .....	2
Setting Up .....	3
Signing Up for AWS .....	3
Creating an IAM User .....	3
Enabling AWS Mobile Hub .....	4
Signing in to Mobile Hub and Creating Your Project .....	4
Getting Started .....	5
App Analytics .....	6
Upgrade your app from Legacy App Analytics to User Engagement .....	6
App Content Delivery .....	7
App Content Delivery At a Glance .....	8
Configuring the App Content Delivery Feature .....	8
Viewing AWS Resources Provisioned for this Feature .....	8
Quickstart App Details .....	11
Cloud Logic .....	12
Cloud Logic At a Glance .....	13
Viewing AWS Resources Provisioned for this Feature .....	13
Quickstart App Details .....	14
Connectors .....	15
Connectors At a Glance .....	16
Microsoft Dynamics .....	17
Configuring Microsoft Dynamics Authorization for Your App .....	17
Microsoft Dynamics Connector API Details .....	18
HubSpot .....	18
Configuring HubSpot Authorization for Your App .....	18
HubSpot Connector API Details .....	18
QuickBooks .....	18
Configuring QuickBooks Authorization for Your App .....	19
QuickBooks Connector API Details .....	19
Marketo .....	19
Configuring Marketo Authorization for Your App .....	19
Marketo Connector API Details .....	19
Salesforce .....	20
Configuring Salesforce Authorization for Your App .....	20
Salesforce Connector API Details .....	20
Zendesk .....	20
Configuring Zendesk Authorization for Your App .....	20
Zendesk Connector API Details .....	21
HTTPS Redirect URLs .....	21
Setting up universal links for iOS apps .....	21
Conversational Bots .....	24
Conversational Bots At a Glance .....	25
NoSQL Database .....	26
NoSQL Database At a Glance .....	27
Learn more .....	27
Example Table Schemas .....	27
Configuring Your Tables .....	28
NoSQL Table Terminology .....	28
Data Permissions .....	28
Retrieving Data .....	30
Learn more .....	30
Quickstart App Details .....	31
Push Notifications .....	32
Push Notifications At a Glance .....	33

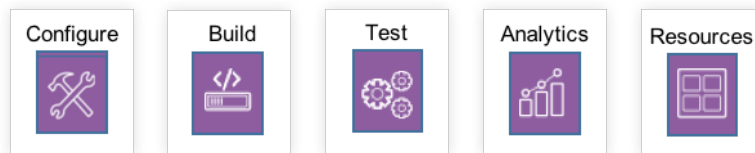
---

Configuring the Push Notifications Feature .....	34
Setting Up Push Notification Services .....	34
Setting Up iOS Push Notification .....	34
Setting Up Android Push Notification .....	41
Viewing AWS Resources Provisioned for this Feature .....	43
Quickstart App Details .....	44
Creating Test Push Notifications for the Quickstart app with Amazon SNS .....	45
User Data Storage .....	46
User Data Storage At a Glance .....	47
Viewing AWS Resources Provisioned for this Feature .....	48
Quickstart App Details .....	49
User Engagement .....	50
User Engagement At a Glance .....	51
User Sign-in .....	52
User Sign-in Feature At a Glance .....	54
Configuring User Sign-in .....	54
User Sign-in Providers .....	55
User Sign-in Requirement .....	56
User Sign-in and AWS Identity and Access Management (IAM) .....	56
Setting Up User Authentication .....	56
Setting Up Facebook Authentication .....	57
Setting Up Google Authentication .....	60
Setting Up Custom Authentication .....	72
Viewing AWS Resources Provisioned for this Feature .....	72
Quickstart App Details .....	73
Tutorial: Use Mobile Hub to Build a Chat App .....	74
Setup and Requirements .....	74
Prerequisites .....	74
Android System Requirements .....	75
iOS System Requirements .....	75
App Features .....	75
App Design and Implementation .....	75
Using the App .....	76
Step 1: Creating a Mobile Hub Project .....	77
Step 2: Adding User Sign-in .....	78
Step 3: Adding NoSQL Database .....	79
Step 4: Adding User Data Storage .....	83
Step 5: Adding Push Notifications .....	84
Step 6: Integrating Backend Features .....	86
Integrating Android Apps .....	87
Integrating iOS Apps .....	89
Sample App Code Tour .....	90
Android Source Code .....	90
iOS Source Code .....	103
Document History .....	114
Reference .....	116
IAM Usage in Mobile Hub .....	116
Controlling Access to Your Mobile Hub Project .....	116
Understanding Mobile Hub Permissions .....	116
Understanding AWS Identity and Access Management .....	116
Controlling Access to Mobile Hub Projects .....	117
Mobile Hub Service Role and Policies .....	118
Authentication and Access Control Basics .....	126
Overview of Access Permissions Management .....	128
Amazon S3 Security Considerations .....	131
Object Lifecycle Management .....	131
Object Encryption .....	131
Object Versioning .....	131

Bucket Logging ..... 131

# What Is AWS Mobile Hub?

---



[AWS Mobile Hub](#) provides an integrated console experience that enables you to quickly create and configure powerful mobile app backend features and integrate them into your mobile app. You create a project by selecting features to add to your app.

The features and AWS services that are supported by Mobile Hub are constantly evolving. Currently they include:

- [App Analytics](#) (p. 6)
- [App Content Delivery](#) (p. 7)
- [Cloud Logic](#) (p. 12)
- [NoSQL Database](#) (p. 26)
- [Push Notifications](#) (p. 32)
- [User Data Storage](#) (p. 46)
- [User Sign-in](#) (p. 52)
- [Connectors](#) (p. 15)
- [Conversational Bots](#) (p. 24)
- [User Engagement](#) (p. 50)

When you build your project for iOS Objective-C, iOS Swift, or Android, Mobile Hub automatically provisions and configures all of the AWS service resources that your app's features require. Mobile Hub then guides you through integrating the features into your app code and downloading a fully working quickstart app project that demonstrates those features.

After your mobile app is built, you can use Mobile Hub to test your app, then monitor and visualize how it is being used.

AWS Mobile Hub enables you to select the region in which your project's resources will be created. For more information about AWS regions, see [Regions and Endpoints](#).

When you use AWS Mobile Hub, you pay only for the underlying services that Mobile Hub provisions based on the features you choose in the Mobile Hub console. For more information, see [Pricing](#).

## How Can I Use AWS Mobile Hub?

Mobile Hub provides all the information you need to use your sample app project:

- Explore the details of AWS mobile features
- Configure AWS services as mobile back ends
- Build your custom app on top of the solid foundation of your Mobile Hub sample app
- Get the app components and functional sample code you need for an app project you build from scratch.

To get started see [Setting Up AWS Mobile Hub \(p. 3\)](#).



# Setting Up AWS Mobile Hub

---

Before you use AWS Mobile Hub for the first time, you must complete the following tasks:

## Topics

- [Signing Up for AWS \(p. 3\)](#)
- [Creating an IAM User \(p. 3\)](#)
- [Enabling AWS Mobile Hub \(p. 4\)](#)

## Signing Up for AWS

To use AWS Mobile Hub, you need an AWS account. Your account has access to all available services, but you are charged only for the services you use. If you are a new AWS customer, you can get started with the [AWS Free Tier](#).

## Creating an IAM User

To provide better security, we recommend that you do not use your AWS root account to access Mobile Hub. Instead, create an AWS Identity and Access Management (IAM) user, or use an existing IAM user, in your AWS account and then access Mobile Hub with that user. For more information, see [AWS Security Credentials](#) in the AWS General Reference. .

If you signed up for AWS but have not created an IAM user for yourself, you can create one by using the IAM console. First, create an IAM administrator group, then create and assign a new IAM user to that group.

### To create an IAM administrators group

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane, choose **Groups**, and then choose **Create New Group**.
3. For **Group Name**, type a name for your group, such as **Administrators**, and then choose **Next Step**.
4. In the list of policies, select the check box next to the **AdministratorAccess** policy. You can use the **Filter** menu and the **Search** box to filter the list of policies.
5. Choose **Next Step**, and then choose **Create Group**. Your new group is listed under **Group Name**.

The following procedure describes how to create an IAM user for yourself, add the user to the administrators group, and create a password for the user.

#### To add an IAM user to your group and assign a password

1. In the navigation pane, choose **Users**, and then choose **Create New Users**.
2. In box **1**, type a user name. Clear the check box next to **Generate an access key for each user**. Then choose **Create**.
3. In the list of users, choose the name (not the check box) of the user you just created. You can use the **Search** box to search for the user name.
4. In the **Groups** section, choose **Add User to Groups**.
5. Select the check box next to the administrators group. Then choose **Add to Groups**.
6. Scroll down to the **Security Credentials** section. Under **Sign-In Credentials**, choose **Manage Password**.
7. Select **Assign a custom password**. Then type a password in the **Password** and **Confirm Password** boxes. When you are finished, choose **Apply**.

## Enabling AWS Mobile Hub

AWS Mobile Hub administers AWS resources for mobile app projects on behalf of the customer. This includes automation that creates AWS Identity and Access Management (IAM) roles for mobile app users and updates their permissions based on the features that are enabled in a mobile app project. Because these operations require administrative privileges (the ability to create and modify IAM roles), only a user with administrative privileges may enable Mobile Hub to do this. These are the steps an administrative user must take in order to enable AWS Mobile Hub in an AWS account. This only needs to be done once.

#### To enable Mobile Hub in an AWS account

1. Navigate to the AWS Mobile Hub console at <https://console.aws.amazon.com/mobilehub/>.
2. Choose **Get Started**.
3. Review the details of the **First things first...** page.
4. Choose **Yes, grant permissions**.

## Signing in to Mobile Hub and Creating Your Project

A Mobile Hub project is a logical workspace that contains the features you choose to incorporate into your mobile app. You can create as many projects as you wish.

#### To create a Mobile Hub project

1. Choose **Get Started** or **Create new project**.
2. For **Project name**, type a name for your project.
3. Choose **Create project**.

# Getting Started with AWS Mobile Hub

---

To get started using AWS Mobile Hub:

- **Read** about the mobile app features Mobile Hub supports, which include: [App Content Delivery](#) (p. 7); [Cloud Logic](#) (p. 12); [NoSQL Database](#) (p. 26); [Push Notifications](#) (p. 32); [User Data Storage](#) (p. 46); [User Sign-in](#) (p. 52); [Connectors](#) (p. 15); [Conversational Bots](#) (p. 24); [User Engagement](#) (p. 50); or see, [What Is AWS Mobile Hub?](#) (p. 1)
- **Try** AWS Mobile Hub (<https://aws.amazon.com/mobile>)

Easy configuration of the AWS services used by common mobile app features, plus working iOS and Android demo app, SDK and integraton helper code customized for your project. Follow the steps at [Setting Up AWS Mobile Hub](#) (p. 3).

- **Create** the [AWSSampleMessenger tutorial](#) (p. 74) app

This walkthrough demonstrates configuring and integrating several Mobile Hub features into a working demonstration chat app for iOS and Android.

- **Check out** the Mobile Blog (<https://aws.amazon.com/blogs/mobile/>)

Recent articles on how to use Mobile Hub for building cloud backend components for common mobile app features.

# App Analytics

---

The App Analytics feature has now been replaced by the Mobile Hub [User Engagement \(p. 50\)](#) feature, which is based on the [Amazon Pinpoint](#) service. Your existing projects that utilize App Analytics and your access to the visualization of your apps' usage metrics in Mobile Analytics will continue to function as before.

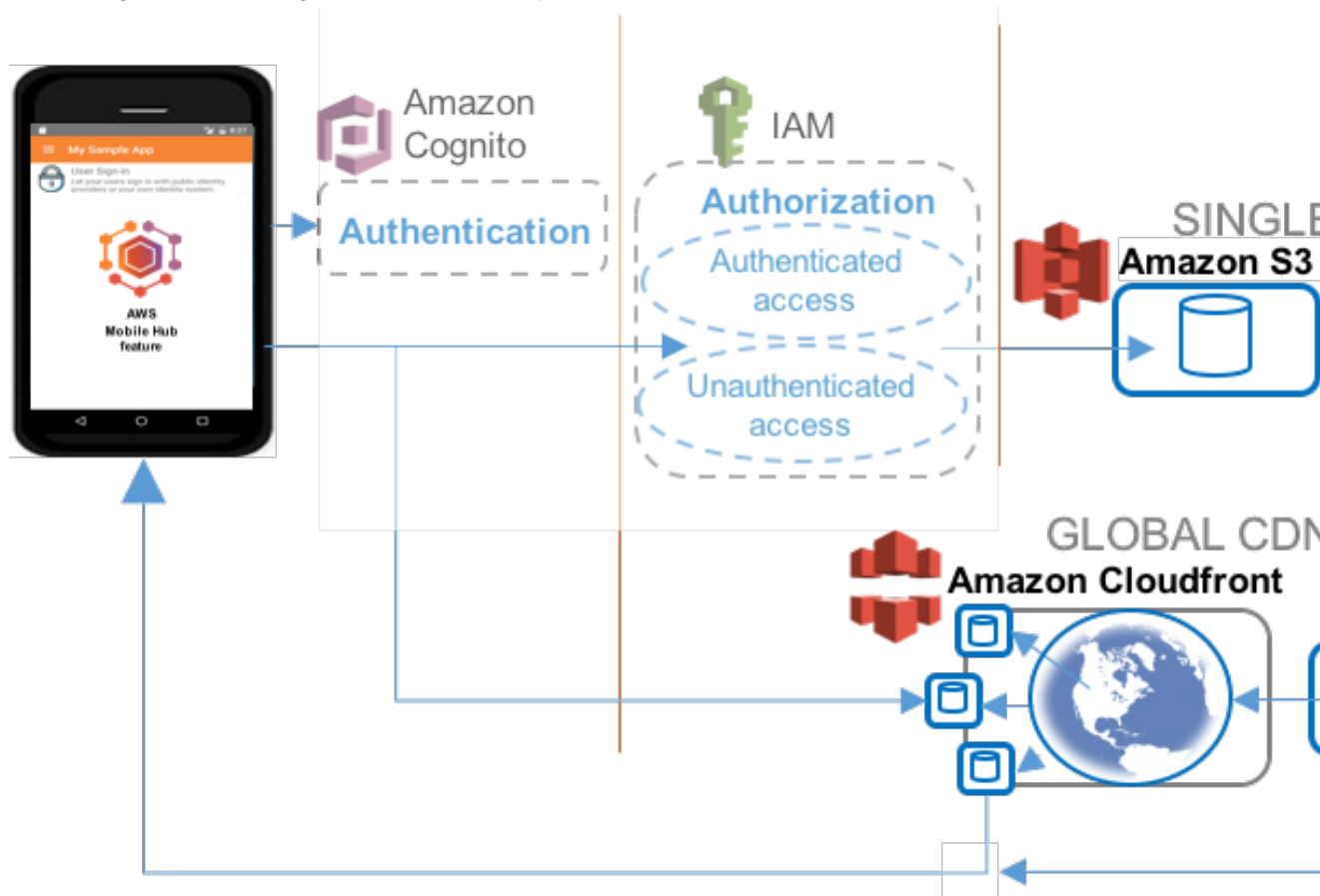
## Upgrade your app from Legacy App Analytics to User Engagement

When you sign in to the Mobile Hub console and open a project for an existing app that uses App Analytics, you will see that the **User Engagement** feature card has replaced the **App Analytics** card, and is marked as enabled to show that your legacy App Analytics feature is still enabled.

Choose the **User Engagement** card, and then choose **Enable engagement** to upgrade App Analytics in your app to add [Amazon Pinpoint](#) campaigns to your app.

# App Content Delivery

Choose AWS Mobile Hub App Content Delivery to add access to cloud content to your mobile app, from a single location or a global Content Delivery Network (CDN).



The App Content Delivery feature enables you to store app assets, like resource or media files, in the cloud so you can download and cache them within your app. Mobile Hub offers two choices for distributing these files: either from a single location using an [Amazon S3](#) bucket or distributed through a global content delivery network by using [Amazon CloudFront](#).

Topics

- [App Content Delivery At a Glance \(p. 8\)](#)
- [Configuring the App Content Delivery Feature \(p. 8\)](#)
- [Viewing AWS Resources Provisioned for this Feature \(p. 8\)](#)
- [Quickstart App Details \(p. 11\)](#)

## App Content Delivery At a Glance

<b>AWS services and resources configured</b>	<ul style="list-style-type: none"><li>• <b>Amazon CloudFront - Content Delivery Network</b> (see <a href="#">Amazon CloudFront</a>) <a href="#">Concepts</a>   <a href="#">Console</a>   <a href="#">Pricing</a></li><li>• <b>Amazon S3 Bucket</b> (see <a href="#">Amazon Simple Storage Service Getting Started Guide</a>) <a href="#">Concepts</a>   <a href="#">Console</a>   <a href="#">Pricing</a></li></ul> <p>Mobile Hub-enabled features use Amazon Cognito for authentication and IAM for authorization. For more information, see <a href="#">User Sign-in (p. 52)</a>.</p> <p>For more information, see <a href="#">Viewing AWS Resources Provisioned for this Feature (p. 8)</a>.</p>
<b>Configuration options</b>	<p>This feature enables the following mobile backend capabilities:</p> <ul style="list-style-type: none"><li>• <b>Single location</b> (AWS storage in a single regional location)</li><li>• <b>Global CDN</b> (AWS storage on a global Content Distribution Network)</li></ul> <p>For more information, see <a href="#">Configuring the App Content Delivery Feature (p. 8)</a>.</p>
<b>Quickstart app demos</b>	<p>This feature adds the following to a quickstart app generated by Mobile Hub:</p> <ul style="list-style-type: none"><li>• View file list in AWS storage, download and view files, and manage their local cache.</li><li>• Same behavior from a single storage location and a global content distribution network.</li></ul>

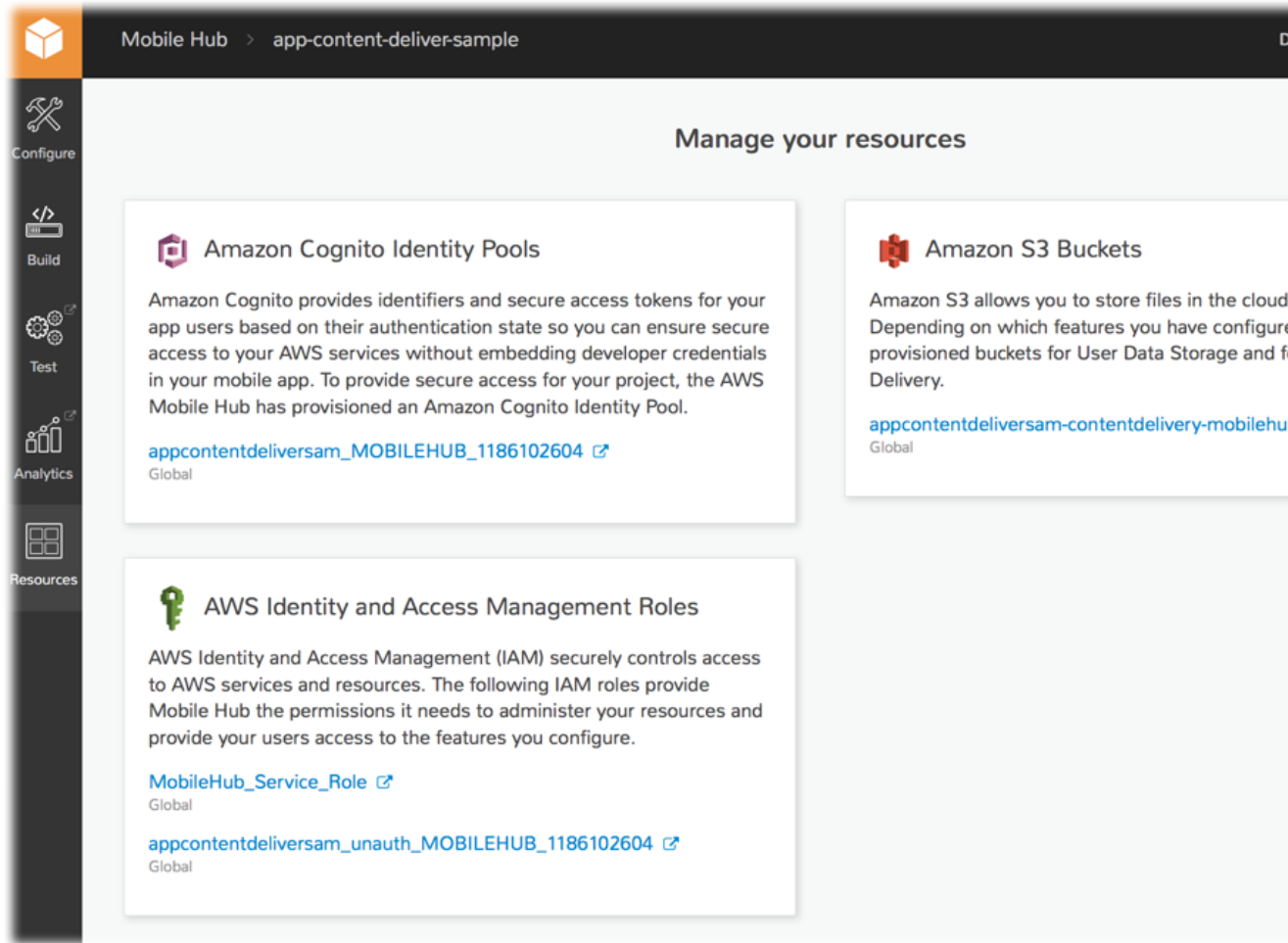
## Configuring the App Content Delivery Feature

If you choose the **Single location** option, Mobile Hub creates an Amazon S3 bucket and pre-populates the bucket with a few sample files that are distributed to your quickstart app directly from there.

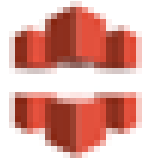
If you choose the **Global CDN** option, Mobile Hub provisions an Amazon CloudFront distribution to deliver files to your app. Amazon CloudFront caches your files using your Amazon S3 bucket as the source (origin) in edge locations around the world to provide faster, lower latency access to your files. Learn more about [Amazon CloudFront](#).

## Viewing AWS Resources Provisioned for this Feature

The following image shows the Mobile Hub **Resources** pane displaying elements typically provisioned for the App Content Delivery feature with **Single location** selected.



The following image illustrates the resource typically provisioned for the additional CloudFront element of the App Content Delivery feature with **Global CDN** selected.



# Amazon CloudFront

## Distributions

Amazon CloudFront is a content delivery web service that provides faster access to your application assets stored in the cloud. Amazon CloudFront can be configured with your App Content Delivery feature.

[d24rcrohqrsr1y.cloudfront.net](https://d24rcrohqrsr1y.cloudfront.net)

Global

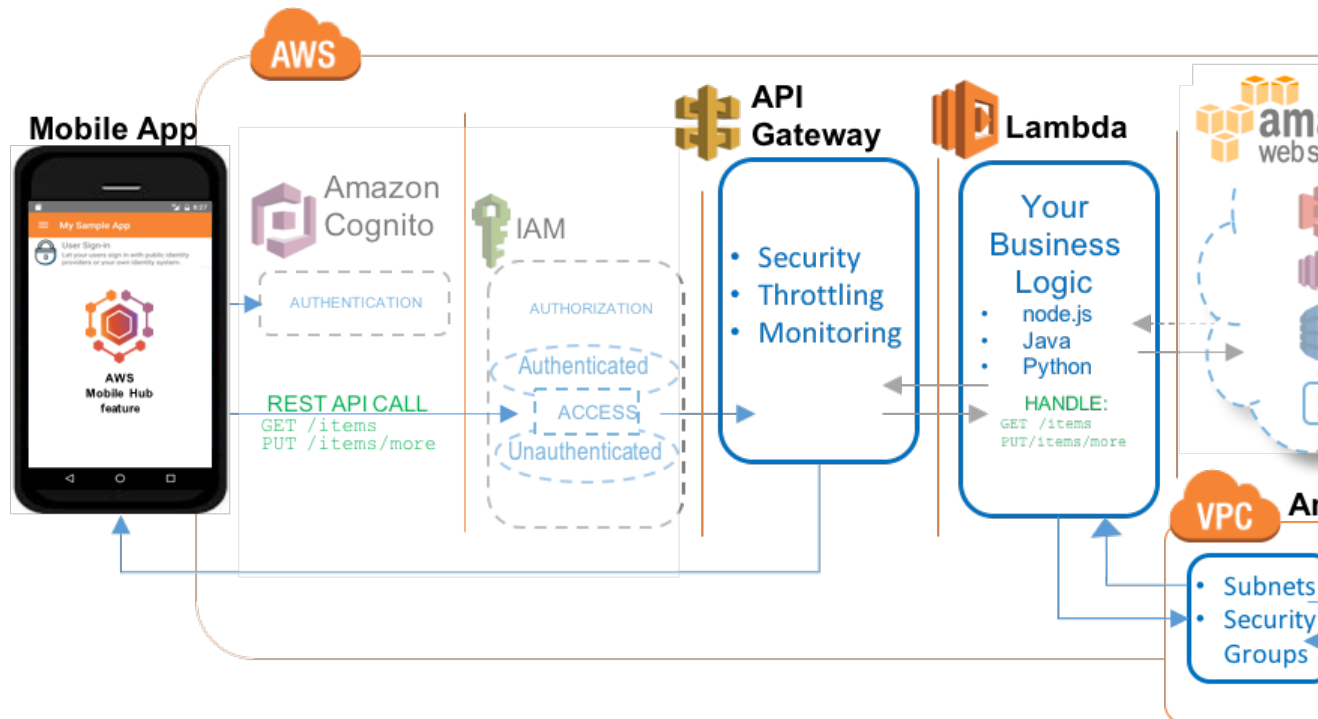


## Quickstart App Details

In the Mobile Hub quickstart app, the App Content Delivery demo lists a set of image files that can be downloaded and cached locally and displayed on the device. The user can also delete the local copy of the image files.

# Cloud Logic Mobile Backend Feature for Your Mobile App

Choose the AWS Mobile Hub Cloud Logic mobile backend service feature to add business logic functions in the cloud and extend to other AWS services for your app, with no cost for server set up or maintenance.



The Cloud Logic feature lets you build backend services using [AWS Lambda](#) functions that you can call from your mobile app. Using Cloud Logic, you can run code in the cloud to process business logic

for your apps and share the same code for both iOS and Android apps. The Cloud logic feature is powered by AWS Lambda functions, which allow you to write code without worrying about managing frameworks and scaling backend infrastructure. You can write your functions in JavaScript, Java, or Python.

Topics

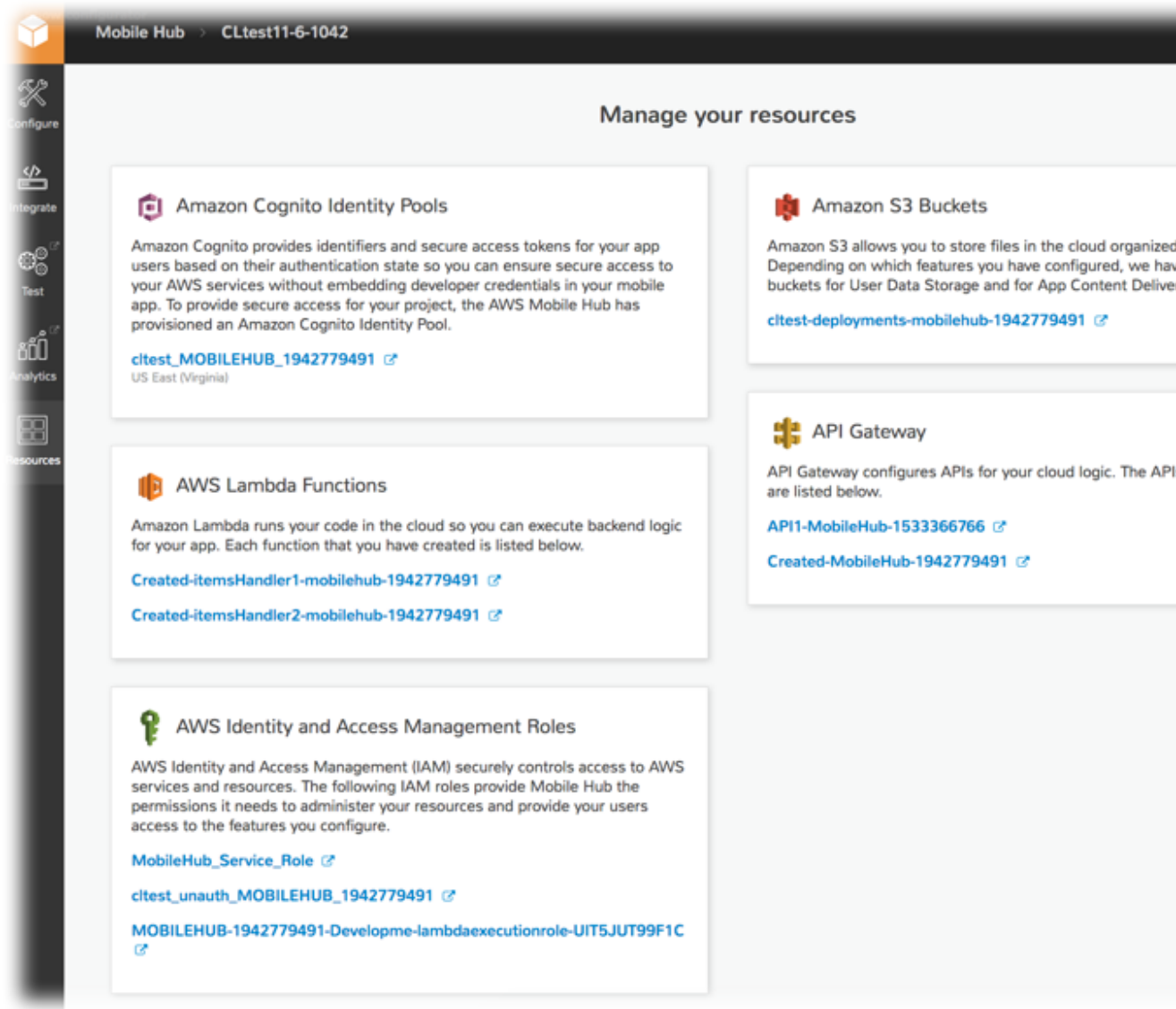
- [Cloud Logic At a Glance \(p. 13\)](#)
- [Viewing AWS Resources Provisioned for this Feature \(p. 13\)](#)
- [Quickstart App Details \(p. 14\)](#)

## Cloud Logic At a Glance

<p><b>AWS services and resources configured</b></p>	<ul style="list-style-type: none"> <li>• <b>Amazon API Gateway</b> (see <a href="#">Amazon API Gateway Developer Guide</a>)  <a href="#">Concepts</a>   <a href="#">Console</a>   <a href="#">Pricing</a></li> <li>• <b>AWS Lambda</b> (see <a href="#">AWS Lambda Developer Guide</a>)  <a href="#">Concepts</a>   <a href="#">Console</a>   <a href="#">Pricing</a></li> <li>• <b>Amazon Virtual Private Cloud</b> (see <a href="#">Amazon VPC User Guide</a>)  <a href="#">Concepts</a>   <a href="#">Console</a>   <a href="#">Pricing</a></li> <li>• <b>AWS CloudFormation</b> (see <a href="#">AWS CloudFormation User Guide</a>)  <a href="#">Concepts</a>   <a href="#">Console</a>   <a href="#">Pricing</a></li> </ul> <p>Mobile Hub-enabled features use Amazon Cognito for authentication and IAM for authorization. For more information, see <a href="#">User Sign-in (p. 52)</a>.</p> <p>For more information, see <a href="#">Viewing AWS Resources Provisioned for this Feature (p. 8)</a>.</p>
<p><b>Configuration options</b></p>	<p>This feature enables the following mobile backend capabilities:</p> <ul style="list-style-type: none"> <li>• Provides a default Hello World Lambda function that accepts the parameter value entered by the app user and returns it back to an app.</li> <li>• Enables you to choose an existing function from the list provided or use the AWS Lambda console to create new functions.</li> </ul>
<p><b>Quickstart app demos</b></p>	<p>This feature adds the following functionality to a quickstart app generated by Mobile Hub:</p> <ul style="list-style-type: none"> <li>• User can specify an AWS Lambda function by name, provide parameters and call a function and see the value returned by the function</li> </ul>

## Viewing AWS Resources Provisioned for this Feature

The following image shows the Mobile Hub **Resources** pane displaying elements typically provisioned for the Cloud Logic feature.



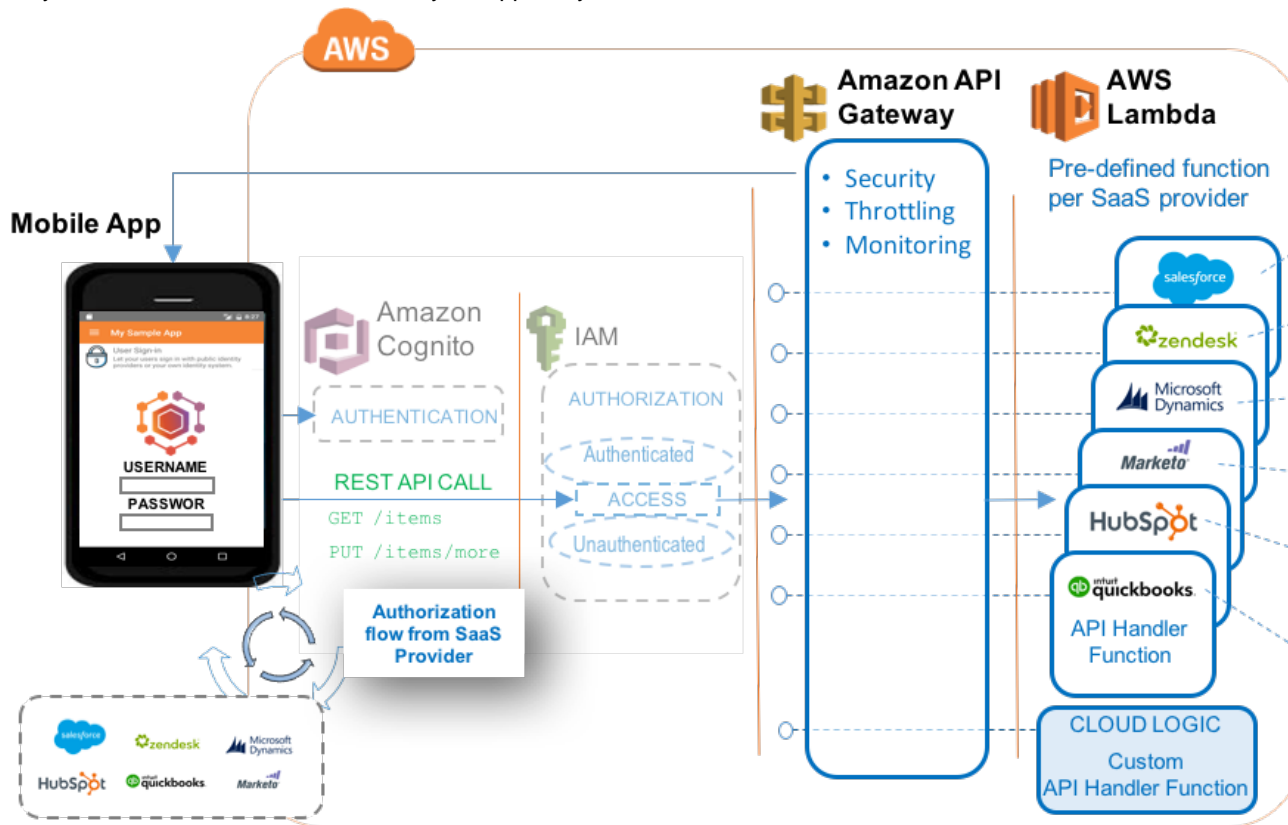
## Quickstart App Details

Your quickstart app includes code to use AWS Lambda APIs to invoke any functions you have selected in your project. Adding Cloud Logic to your quickstart app provides a Hello World default Lambda function. You can also choose an existing Lambda function from your AWS account, or you can create a new one. When you choose the edit button, you are taken to the function editor in the AWS Lambda console. From the Lambda console, you can edit the code directly or upload a package of source and libraries as a .zip file.

In the demo screen of the Cloud Logic quickstart app, you can enter the name and input parameters of the Lambda function you wish to invoke. The quickstart app then calls your Lambda function and displays the results it returns.

# Connectors

Choose the AWS Mobile Hub Connectors mobile backend service feature to connect your iOS and Android mobile apps to the SaaS platform you use. Using REST APIs, gain visibility into and control over your SaaS traffic. Lower the cost of your app lifecycle.



AWS Mobile Hub Connectors are AWS APIs that are pre-defined to enable you to rapidly develop mobile apps that connect to your application on a SaaS platform. Connectors optimize and simplify connecting your app to your SaaS application in several ways. The underlying SaaS APIs are normalized, which provides you a consistent object model, and paging and filtering behaviors.

Connectors provide a central point for auditing and metering API activity, for testing, and for enabling caching and throttling. That adds up to savings in time and developer and administrator costs throughout an app's lifecycle and an increase in code reusability.

Like Cloud Logic, Connectors use an Lambda function exposed to your app as a REST API by Amazon API Gateway. That means that the required AWS services and permissions are provisioned for you in minutes. You can download of a native SDK for both iOS and Android as well as a quickstart app demonstrating your provisioned services on iPhone and Android.

Unlike Cloud Logic, we author and maintain the business logic of a Connector for you. As minor SaaS platform updates happen we will adjust behind the scenes, where possible, so that you can avoid the need to republish your app.

Enabling Connectors in your project will also enables the [User Data Storage \(p. 46\)](#) feature. We take this step to provide the infrastructure for secure file transfer, as this is a common requirement in SaaS application scenarios.

The SaaS providers currently supported are:



[Microsoft Dynamics Connector for Mobile Hub \(p. 17\)](#)



[HubSpot Connector for Mobile Hub \(p. 18\)](#)



[Marketo Connector for Mobile Hub \(p. 19\)](#)



[QuickBooks Connector for Mobile Hub \(p. 18\)](#)



[Salesforce Connector for Mobile Hub \(p. 20\)](#)



[Zendesk Connector for Mobile Hub \(p. 20\)](#)

## Connectors At a Glance

<b>AWS services</b>	<ul style="list-style-type: none"><li>• <b>Amazon API Gateway</b> (see <a href="#">Amazon API Gateway Developer Guide</a>) <a href="#">Concepts</a>   <a href="#">Console</a>   <a href="#">Pricing</a></li></ul>
---------------------	---

<b>and resources configured</b>	<ul style="list-style-type: none"><li>• <b>AWS Lambda</b> (see <a href="#">AWS Lambda Developer Guide</a>) <a href="#">Concepts</a>   <a href="#">Console</a>   <a href="#">Pricing</a></li><li>• <b>AWS CloudFormation</b> (see <a href="#">AWS CloudFormation User Guide</a>) <a href="#">Concepts</a>   <a href="#">Console</a>   <a href="#">Pricing</a></li></ul> <p>Mobile Hub-enabled features use Amazon Cognito for authentication and IAM for authorization. For more information, see <a href="#">User Sign-in (p. 52)</a>.</p> <p>For more information, see <a href="#">Viewing AWS Resources Provisioned for this Feature (p. 48)</a>.</p>
<b>Configuration options</b>	<p>This feature enables the following mobile backend capabilities:</p> <ul style="list-style-type: none"><li>• Create and configure APIs for your project that connect your app to the SaaS provider you use. For configuration details for each connector see:<ul style="list-style-type: none"><li>• HubSpot</li><li>• Marketo</li><li>• Microsoft Dynamics</li><li>• Quickbooks</li><li>• Salesforce</li><li>• Zendesk</li></ul></li><li>• Integrate your app by downloading the example of the quickstart app, and a package of native iOS and Android SDKs plus helper code, all of which are dynamically generated to match your Mobile Hub project.</li></ul>
<b>Quickstart app demos</b>	<p>This feature adds the following functionality to a quickstart app generated by Mobile Hub:</p> <ul style="list-style-type: none"><li>• Prompts the user to log in to the SaaS provider and performs the authorization flow</li><li>• Create, read, update, and delete objects (ie. accounts, tickets, or contacts) within the SaaS backend</li></ul>

## Microsoft Dynamics Connector for Mobile Hub



The Microsoft Dynamics Connector implements all the methods and properties available when directly calling Microsoft Dynamics APIs.

### Configuring Microsoft Dynamics Authorization for Your App

To enable your app to use OAuth authentication for Microsoft Dynamics user validation, register your app with Azure Active Directory cloud services. To register, create an account at [portal.azure.com](https://portal.azure.com), and then create an application in their system.

To configure your Connector backend to use your Microsoft Dynamics app registration, provide the following configuration values from [portal.azure.com](https://portal.azure.com).

- **Dynamics Application ID** (OAuth Client ID)

- **Dynamics Redirect URL** (OAuth Redirect URI)
- **Dynamics Resource URI**

The Resource URI is in the form of: `your_dynamics_subdomain.crm.dynamics.com`.

## Microsoft Dynamics Connector API Details

See the [Microsoft Dynamics Connector API Reference](#) for more details.

## HubSpot Connector for Mobile Hub



The HubSpot Connector implements all the methods and properties available when directly calling HubSpot APIs.

### Configuring HubSpot Authorization for Your App

To enable your app to use OAuth authentication for HubSpot user validation, register your app with their cloud services. To register, create a HubSpot account at <https://app.hubspot.com>, and then create an application in their system.

To find information about HubSpot accounts, applications, and OAuth validation, see [HubSpot OAuth2 Authentication](#).

To configure your Connector backend to use your HubSpot app registration, provide the following configuration values from <https://app.hubspot.com>.

- **HubSpot Client ID** (OAuth Client ID)
- **HubSpot Redirect URL** (OAuth Redirect URI)

HubSpot Redirect URLs must use the HTTPS scheme. To learn about recommended coding practices to meet this requirement, see [Using HTTPS OAuth redirect URLs \(p. 21\)](#).

- **HubSpot Portal ID**

The Portal ID is available in the upper right corner of the portal page when logged in.

### HubSpot Connector API Details

See the [HubSpot Connector API Reference](#) for more details.

## QuickBooks Connector for Mobile Hub



The QuickBooks Connector implements all the methods and properties available when directly calling QuickBooks APIs.



## Configuring QuickBooks Authorization for Your App

To enable your app to use OAuth authentication for QuickBooks user validation, register your app with their cloud services. To register, create a QuickBooks account at <https://developer.intuit.com>, and then create an application in their system.

To find information about QuickBooks accounts, applications, and OAuth validation, see <https://developer.intuit.com>.

To configure your Connector backend to use your QuickBooks app registration, provide the following configuration values from <https://developer.intuit.com>.

- **QuickBooks OAuth Consumer Key** (OAuth Client ID)
- **Quickbooks OAuth Callback URL** (OAuth Redirect URI)

## QuickBooks Connector API Details

See the [QuickBooks Connector API Reference](#) for more details.

## Marketo Connector for Mobile Hub



The Marketo Connector implements all the methods and properties available when directly calling Marketo APIs.

## Configuring Marketo Authorization for Your App

To enable your app to use OAuth authentication for Marketo user validation, register your app with their cloud services. To register, create a Marketo account at <https://login.marketo.com>, and then create an application in their system.

To find information about Marketo accounts, applications, and OAuth validation, see [Marketo REST API documentation](#).

To view the values, you will need to configure your Marketo Connector using Marketo credentials assigned a role that has access to developer resources. Your app should provide Marketo credentials (ClientID and Secret) on a per user basis.

To configure your Connector backend to use your Marketo app registration, provide the following configuration values from <https://app.marketo.com>.

- **Marketo REST API Endpoint** (OAuth Client ID)

This URI should be in the form of: `https://xxx-xxx-xxx.mktorest.com/rest.`

- **Marketo REST API Identity** (OAuth Redirect URI)

This URI should be in the form of: `https://xxx-xxx-xxx.mktorest.com/identity.`

## Marketo Connector API Details

See the [Marketo Connector API Reference](#) for more details.

## Salesforce Connector for Mobile Hub



The Salesforce Connector implements all the methods and properties available when directly calling Salesforce APIs.

### Configuring Salesforce Authorization for Your App

To enable your app to use OAuth authentication for Salesforce user validation, register your app with their cloud services. To register, create a Salesforce account at <https://login.salesforce.com>, and then create an application in their system.

To learn more about Salesforce accounts, applications, and OAuth validation, see <https://developer.salesforce.com/>

To configure your Connector backend to use your Salesforce app registration, provide the following configuration values from <https://login.salesforce.com/>.

- **Salesforce Consumer Key** (OAuth Client ID)
- **Salesforce Callback URL** (OAuth Redirect URI)

### Salesforce Connector API Details

See the [Salesforce Connector API Reference](#) for more details.

## Zendesk Connector for Mobile Hub



The Zendesk Connector implements all the methods and properties available when directly calling Zendesk APIs.

### Configuring Zendesk Authorization for Your App

To enable your app to use OAuth authentication for Zendesk user validation, register your app with their cloud services. To register, create a Zendesk account at <https://developer.zendesk.com>, then create an application in their system.

To configure your Connector backend to use your Zendesk app registration, provide the following configuration values from <https://developer.zendesk.com>.

- **Zendesk Unique Identifier** (OAuth Client ID)
- **Zendesk Redirect UR** (OAuth Redirect URI)

Zendesk Redirect URLs must use the HTTPS scheme. To learn about recommended coding practices to meet this requirement, see [Using HTTPS OAuth redirect URLs \(p. 21\)](#).

- **Zendesk Subdomain**

## Zendesk Connector API Details

See the [Zendesk Connector API Reference](#) for more details.

## Using HTTPS OAuth redirect URLs

The SaaS providers HubSpot and Zendesk require that the OAuth2 redirect URL for your Connector must use the HTTPS scheme. To avoid a network call, we recommend the following:

- **Projects targeted at the Android platform only**

If you are using your Mobile Hub project to create only an Android version of your app, your Redirect URL can be in the form of `https://localhost/. . .`

- **Projects targeted at the iOS platform only**

If you are using your Mobile Hub project to produce an iOS version of your app, use the following steps to provide a [universal link](#) as the redirect URL.

- **Projects targeted at both iOS and Android platforms**

If you are using your Mobile Hub project to produce both iOS and Android versions of your app:

- For iOS (iOS 9 and above), use the following steps to provide a [universal link](#) as the redirect URL.
- For Android (API 6.0 and above), using the same domain as you configure for your iOS universal link, declare a web site association for your redirect URL using Intents as described in the [Android App Links](#) documentation.

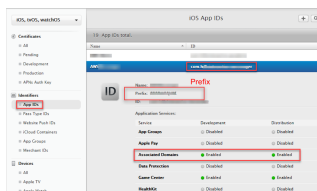
If you encounter issues with your Zendesk HTTPS OAuth Redirect URL using universal links, try using Apple URL Schemes. Set up an HTTPS website containing an URL that redirects back to your app using the URL scheme. Use the URL of that file as your Zendesk Redirect URL. You can use a protected Amazon S3 bucket as the endpoint. It can be created as part of your project by enabling User Data Storage feature, but this step should be taken before enabling your Connector so that you can use the bucket URL as the configuration value for your Connector.

## Setting up universal links for iOS apps

This section describes how to set up iOS universal links that can be used for HTTPS redirect URLs for Mobile Hub SaaS Connectors.

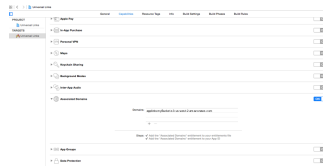
### To set up universal links for iOS

1. Navigate to the S3 console at <https://console.aws.amazon.com/s3/>, choose **Create Bucket** and supply the name and region.
2. Enable **Associated Domains** for your **App ID** in the [Apple Developer Portal](#) by logging into the developer portal, selecting **Certificates, IDs & Profiles**, selecting **App IDs**, and then selecting your app from the list.



3. In Apple Developer Portal, copy your App ID **Prefix** value and save it for later when you create the `apple-app-site-association` file.

4. In Xcode, choose **App Target**, then choose **Capabilities**, then change the **Associated Domains** switch to **ON**



5. In the **Domains:** field of **Associated Domains**, add your S3 bucket subdomain appended to applinks. As an example, if your bucket name is `myBucket` and the region it is created in is `us-west-2`, then your **Associated Domains Domain** would be:

```
applinks:myBucket.s3-us-west-2.amazonaws.com
```

6. Configure your app to handle **universal links** by adding the following `application:continueUserActivity:restorationHandler` to your AppDelegate class.

[Swift]

```
func application(application: UIApplication, continueUserActivity
  userActivity: NSUserActivity, restorationHandler: ([AnyObject]?) -> Void)
  -> Bool {
    if userActivity.activityType == NSUserActivityTypeBrowsingWeb {
      let url = userActivity webpageURL!
      //handle url stuff
    }
    return true
  }
}
```

7. Create a new file named `apple-app-site-association`. The file will contain JSON but its name does not use the `.json` extension.

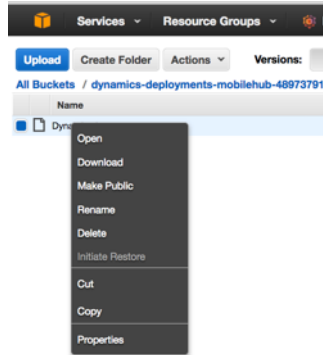
Paste the following JSON object into the file and modify the `appID` value `<your AppPrefix.AppID>` by replacing it with your **AppID** appended to your **App Identifier Prefix**.

For example if your App Prefix is `GR5IUEHB6E` and your App ID is `com.company.appname` then the value of `appID` should be:

```
GR5IUEHB6E.com.company.appname
```

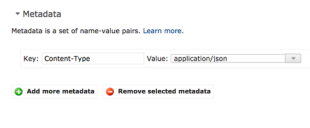
```
{
  "applinks": {
    "apps": [ ],
    "details": [
      {
        "appID": "<your AppPrefix.AppID>",
        "paths": [ "*" ]
      }
    ]
  }
}
```

8. In the S3 console, upload the modified `apple-app-site-association` file to the root of your S3 bucket.
9. a. In the S3 console, right-click your uploaded file and select **make public**.



- b. In the S3 console, open **Properties** for the uploaded file.

In the **Metadata** dropdown, modify **Value**, to `application/json`.



10. Run the app.

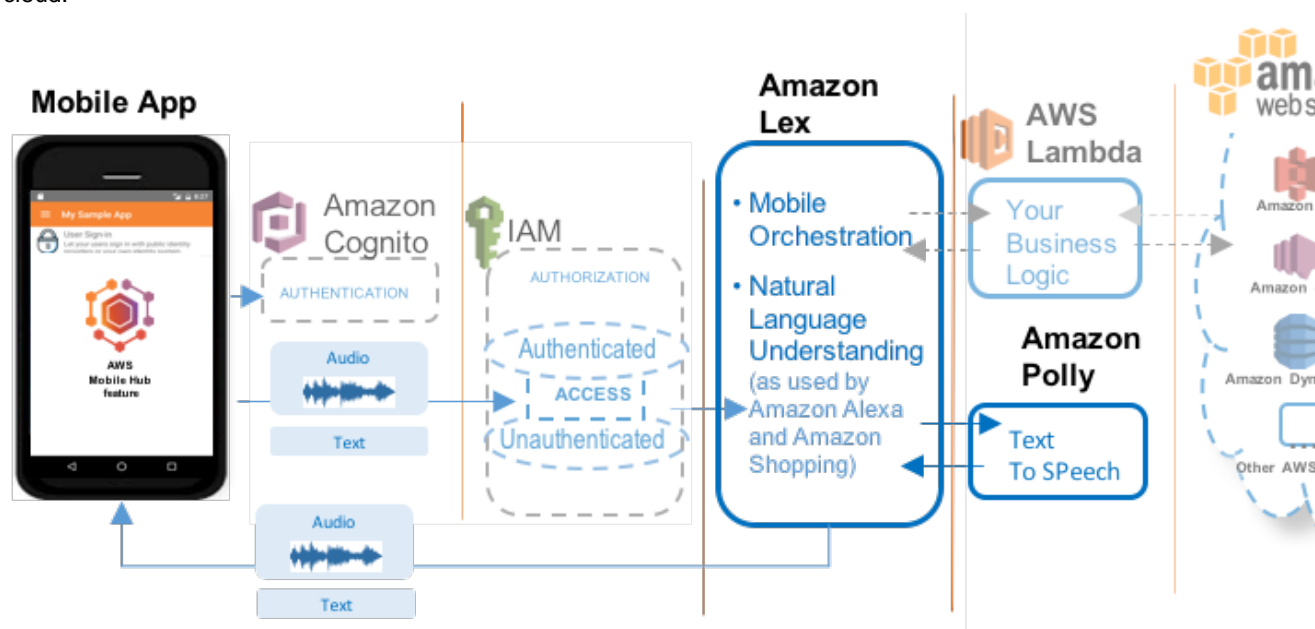
You can test the universal link functionality by sending a link with the URL specified in associated domains to your email or SMS. Selecting the link on the device with the app installed will open the app and the apps restoration handler function in the AppDelegate will manage the link URL and any attributes.

#### Universal Link Caveats:

1. Simply pasting a link into the address bar of Safari will not launch the app as you might expect. Try clicking in the link from SMS or mail client.
2. Universal links are supported for iOS 9+. Universal links in early versions of 9 work only on the device and not the simulator.

# Conversational Bots

Choose the AWS Mobile Hub conversational bots mobile backend service feature to add voice and text natural language understanding interface to your app. Integrate these with your business logic in the cloud.



AWS Mobile Hub conversational bots bring your mobile app the same natural language understanding and business logic integration that power the Amazon Alexa and Amazon Shopping voice and text conversation experiences.

Mobile Hub conversational bots use Amazon Lex, an AWS service for building voice and text conversational interfaces into applications. Amazon Lex has built-in integration with Lambda.

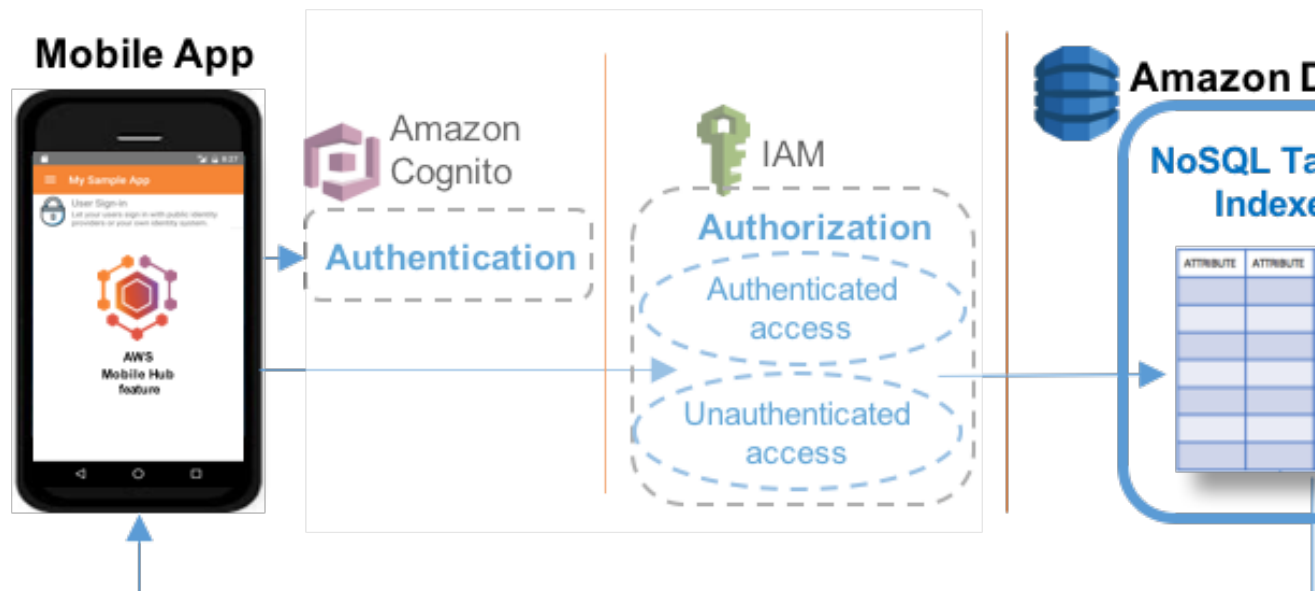
With conversational bots and Amazon Lex, no deep learning expertise is necessary. Specify the basic conversation flow in the Amazon Lex console to create a bot. The service manages the dialogue and dynamically adjusts the responses in the conversation. Using Mobile Hub conversation bots, you can provision and test bots based on demonstration templates or bots you have created in the Amazon Lex console. Mobile Hub provides integration instructions and customized components for reusing the sample app code we generate in your own app.

## Conversational Bots At a Glance

<b>AWS services and resources configured</b>	<ul style="list-style-type: none"><li>• <b>Amazon Lex</b> (see <a href="#">Amazon Lex Developer Guide</a>) <a href="#">Concepts</a>   <a href="#">Console</a></li></ul> <p>Mobile Hub-enabled features use Amazon Cognito for authentication and IAM for authorization. For more information, see <a href="#">User Sign-in (p. 52)</a>.</p>
<b>Configuration options</b>	<p>This feature enables the following mobile backend capabilities:</p> <ul style="list-style-type: none"><li>• Create and configure conversational bots in the Amazon Lex service based on provided demonstration templates or by using the Amazon Lex console to add your customized text and/or speech interactions to your app.</li><li>• Integrate your app by downloading and reusing the code of the quickstart app, a package of native iOS and Android SDKs, plus helper code and on line guidance, all of which are dynamically generated to match your Mobile Hub project.</li></ul>
<b>Quickstart app demos</b>	<p>This feature adds the following functionality to a quickstart app generated by Mobile Hub:</p> <ul style="list-style-type: none"><li>• Enables user to interact with a conversational bot that interacts with Amazon Lex.</li></ul>

# NoSQL Database

Choose the Mobile Hub NoSQL Database mobile backend feature to your mobile app to add database capabilities that are easy to develop and provide scalable performance and cost.



The NoSQL Database feature uses [Amazon DynamoDB](#) to enable you to create database tables that can store and retrieve data for use by your apps.

NoSQL databases are widely recognized as the method of choice for many mobile backend solutions due to their ease of development, scalable performance, high availability, and resilience. For more information, see [From SQL to NoSQL](#) in the *Amazon DynamoDB Developer Guide*.

## Topics

- [NoSQL Database At a Glance](#) (p. 27)
- [Configuring the NoSQL Database Feature](#) (p. 27)
- [Configuring Your Tables](#) (p. 28)
- [Viewing AWS Resources Provisioned for this Feature](#) (p. 30)
- [Quickstart App Details](#) (p. 31)



## NoSQL Database At a Glance

<b>AWS services and resources configured</b>	<ul style="list-style-type: none"><li>• <b>Amazon DynamoDB tables</b> (see <a href="#">Working with Tables in DynamoDB</a>) <a href="#">Concepts</a>   <a href="#">Console</a>   <a href="#">Pricing</a></li></ul> <p>Mobile Hub-enabled features use Amazon Cognito for authentication and IAM for authorization. For more information, see <a href="#">User Sign-in</a> (p. 52).</p> <p>For more information, see <a href="#">Viewing AWS Resources Provisioned for this Feature</a> (p. 30).</p>
<b>Configuration options</b>	<p>This feature enables the following mobile app backend capabilities:</p> <p><a href="#">Configuring Your Tables</a> (p. 28) - Using custom schema, based on a sample schema provided, or by using a wizard that guides you through choices while creating a table</p> <p><a href="#">Data Permissions</a> (p. 28) - Access to your app's data can be:</p> <ul style="list-style-type: none"><li>• <b>Public</b> (enables any mobile app user to read or write any item in the table)</li><li>• <b>Protected</b> (enables any mobile app user to read any item in the table but only the owner of an item can update or delete it)</li><li>• <b>Private</b> (enables only the owner of an item to read and write to a table)</li></ul> <p>For more information, see <a href="#">Configuring the NoSQL Database Feature</a> (p. 27).</p>
<b>Quickstart app demos</b>	<p>This feature adds the following to a quickstart app generated by Mobile Hub:</p> <ul style="list-style-type: none"><li>• Insert and remove sample data, based on the schema you specify in the console.</li><li>• Perform and see the results of NoSQL operations on tables including Get, Scan, and all the example queries displayed by the console as you make design selections.</li></ul>

## Configuring the NoSQL Database Feature

This section describes steps and options for configuring NoSQL Database features in Mobile Hub.

### To add the NoSQL Database feature to your Mobile Hub project

1. Choose **Enable NoSQL**.
2. Choose **Add a new table**.
3. Choose the initial schema for the table. You can use a provided example schema, or generate a schema through the wizard.

## Example Table Schemas

AWS Mobile Hub provides a set of example table schemas for typical mobile apps. If you create a table using one of the example schema templates, the table initially has a set of attributes specific to each example. You can choose one of these templates as the starting schema for your table:

- **News**, which stores author, title, article content, keywords, and other attributes of news articles.

- **Locations**, which stores names, latitude, and longitude of geographic locations.
- **Notes**, which stores private notes for each user.
- **Ratings**, which stores user ratings for a catalog of items.
- **Graffiti Wall**, which stores shared drawing items.

### To add a table using one of the example schema templates in your Mobile Hub project

1. Choose the example template to use for the initial schema of the table.
2. Type a new name in **Table name** to rename the table if you wish. Each template gives the table a default name matching the name of the template.
3. Choose **Public**, **Protected**, or **Private** permissions to grant to the mobile app users for the table. For more information, see [Data Permissions \(p. 28\)](#).
4. (Optional) Under **What attributes do you want on this table?**, you can add, rename, or delete table attributes.
5. (Optional) Choose **Add index** to add **name**, **partition key**, and (optionally) **sort key** for a secondary index for your table.
6. Choose **Create table**.

## Configuring Your Tables

This section describes options for configuring DynamoDB NoSQL tables for your app.

### Contents

- [NoSQL Table Terminology \(p. 28\)](#)
- [Data Permissions \(p. 28\)](#)
  - [Enforcing Permissions \(p. 29\)](#)
  - [Restricting Permissions for Multiple Writers \(p. 29\)](#)
  - [Table Permissions Options in Mobile Hub \(p. 29\)](#)
- [Retrieving Data \(p. 30\)](#)

## NoSQL Table Terminology

Similar to other database management systems, DynamoDB stores data in tables. A table is a collection of data with the following elements.

### Items

Each table contains multiple items. An item is a group of attributes that is uniquely identifiable among all of the other items. Items are similar to rows, records, or tuples in relational database systems.

### Attributes

Attributes are the columns in a DynamoDB table. The rows of the table are the individual records you add, update, read, or delete as necessary for your app.

The table schema provides a set of initial attributes based on the needs of each example. You can remove any of these attributes by choosing **Remove**. If you remove the partition key attribute, then you must designate another attribute as the partition key for the primary index of the table.

You can choose **Add attribute** to add a blank attribute to the table. Give the attribute a name, choose the type of data it will store, and choose whether the new attribute is the partition key or the sort key.

## Indexes

Each table has a built-in primary index, which has a partition key and may also have a sort key. This index allows specific types of queries. You can see the types of queries the table can perform by expanding the **Queries this table can perform** section. To enable queries using other attributes, create additional secondary indexes. Secondary indexes enable you to access data using a different partition key and optional sort key from those on the primary index.

## Data Permissions

When you create a new table, you must set permissions that determine which mobile app users can read and/or write the table's data. You can set these permissions to control access to each table as: public, protected, or private.

## Enforcing Permissions

Use the settings in the **What permissions would you like for this table?** section to enable your mobile app to directly access your NoSQL tables in the Amazon DynamoDB service. Because there is no middle layer between the mobile app and the database service, it is important that you use an appropriate access policy to restrict access to your tables. When you choose permissions for each table, Mobile Hub provisions a fine-grained access control policy for your mobile app users. If you select **Protected** or **Private**, then every operation that is attempted on an item in your table will first check if the **userId** field in the table item (or row) matches the user's Amazon Cognito Identity.

As the value of the primary partition key of a restricted NoSQL Database table will contain the Amazon Cognito Identity of the app user whose action created the item, the key must be called **userId** and be of type **string**). The name and data type of secondary indexes for restricted tables must follow the same pattern: **'userId' (string)**.

## Restricting Permissions for Multiple Writers

After Mobile Hub provisions access restrictions for your tables with **Protected** or **Private** permissions, IAM ensures that only the mobile app user whose action creates an item in the table will be able to write to the attribute values of that item. To design your schema for the case where multiple users need to write data to an existing item, one strategy is to structure your schema in a way that users write to different tables. In this design, the app queries both tables to join data.

For example, customers may create orders in an **order** table and delivery service drivers may write delivery tracking information to a **deliveries** table, where both tables have secondary indexes that allow fast lookup based on **orderId** or **customerId**.

## Table Permissions Options in Mobile Hub

When you create a new table, you must set the table's permissions. These permissions determine who can read data from and who can write data to the table. Mobile Hub offers the following table permissions configurations.

### Public

Public permissions allow all mobile app users to read or write all items in the table. There is no restriction on how you configure the partition key.

### Protected

Protected permissions allow all mobile app users to read all items in the table but only the owner of an item can update or delete it. These permissions grant full access to retrieve data from the table but limited access to update or remove existing items.

Only app users with an Amazon Cognito Identity ID matching the item's partition key can write to the item. The partition key for the table must follow the pattern of **'userId' (string)** value.

#### Private

Private permissions allow only the owner of an item to read and write to it. This enforces the most restrictive set of permissions for accessing the table; however, these permissions offer a higher degree of protection by limiting access.

Only app users with an Amazon Cognito Identity ID matching the item's partition key can read or write to the item. The partition key for the table and for any secondary indexes must follow the pattern of **'userId' (string)**value.

## Retrieving Data

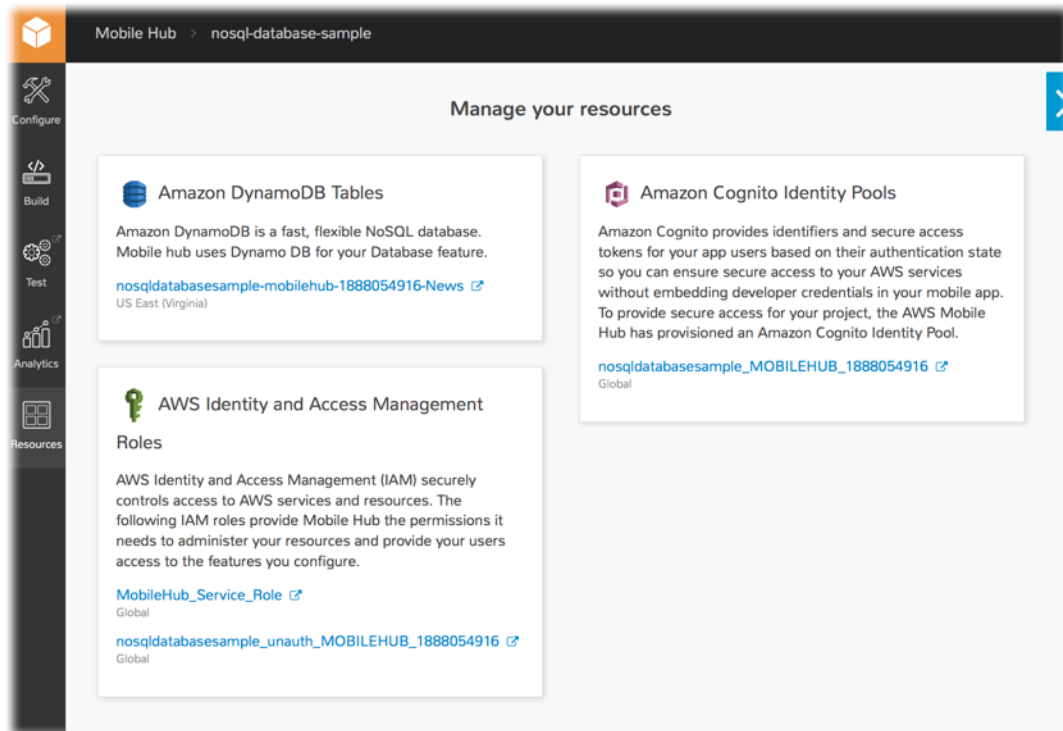
The operations you can use to retrieve data from your NoSQL database include the following:

- `Get`, which retrieves a single item from the table based on matching the primary key.
- `Query`, which finds items in a table or a secondary index using only primary key attribute values.
- `Scan`, which reads every item in a table or secondary index. By default, a `Scan` operation returns all of the data attributes for every item in the table or index. You can use `Scan` to return only some attributes, rather than all of them.
- `Query with Filters`, which performs a `Query` but returns results that are filtered based on a filter expression you create.
- `Scan with Filters`, which performs a `Scan` but returns results that are filtered based on a filter expression you create.

For more information, see [Query and Scan Operations in DynamoDB](#).

## Viewing AWS Resources Provisioned for this Feature

The following image shows the Mobile Hub **Resources** pane displaying the AWS elements typically provisioned for the NoSQL Database feature:



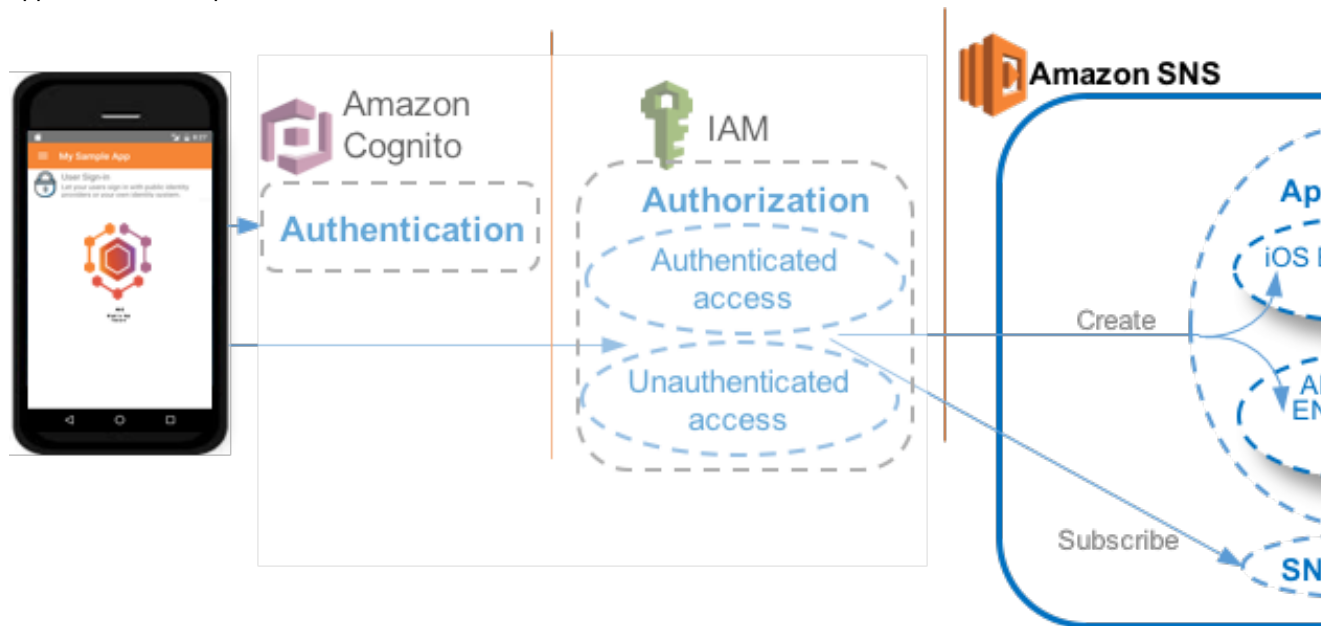
## Quickstart App Details

In the Mobile Hub quickstart app, the NoSQL Database demo shows a list of all tables created during app configuration. Selecting a table shows a list of all queries that are available for that table, based on the choices made regarding its primary indexes, secondary indexes, and sort keys. Tables that you make using the example templates enable an app user to insert and remove sample data from within the app.

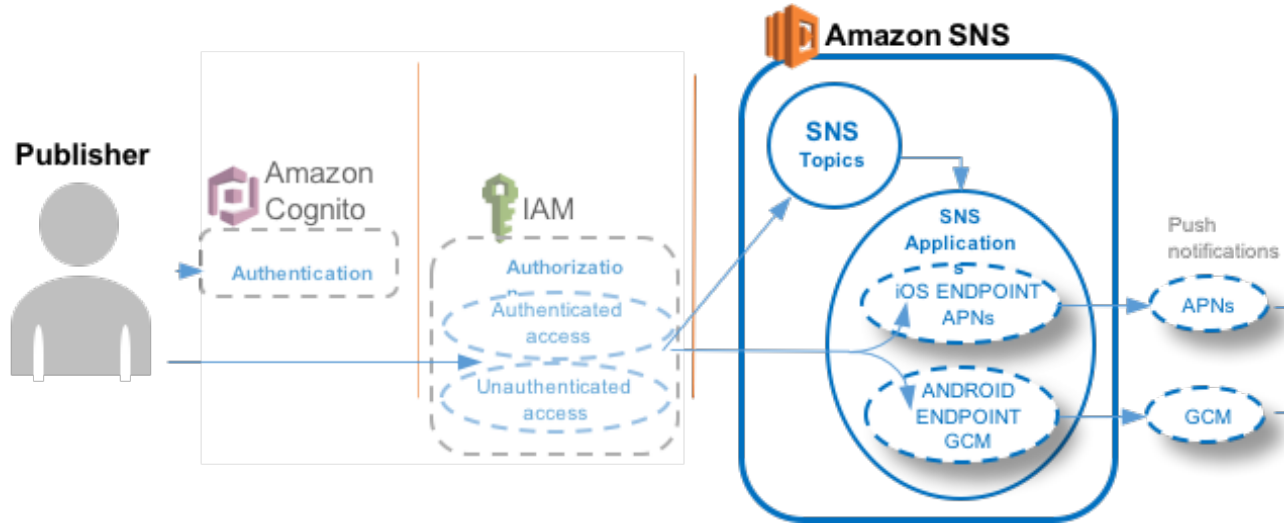
# Push Notifications

Choose AWS Mobile Hub Push Notifications mobile backend feature to add mobile messaging to your mobile app.

The following image shows client perspective on subscription of a mobile app to Amazon SNS applications and topics.



The following image shows the server side developer perspective on publishing of push notifications to devices subscribed to Amazon SNS applications and topics.



The Push Notifications feature enables you to send push notification messages to your iOS and Android apps using [Amazon Simple Notification Service \(Amazon SNS\)](#). You can integrate with Apple and Google messaging services by providing credentials that are provided by those services. You then can send messages directly to individual devices, or publish messages to the SNS topics that installed apps are subscribed to.

Topics

- [Push Notifications At a Glance](#) (p. 33)
- [Configuring the Push Notifications Feature](#) (p. 34)
- [Setting Up Push Notification Services](#) (p. 34)
- [Viewing AWS Resources Provisioned for this Feature](#) (p. 43)
- [Quickstart App Details](#) (p. 44)

## Push Notifications At a Glance

<p><b>AWS services and resources configured</b></p>	<ul style="list-style-type: none"> <li>• <b>Amazon SNS</b> (see <a href="#">Amazon Simple Notification Service Developer Guide</a>)  <a href="#">Concepts</a>   <a href="#">Console</a>   <a href="#">Pricing</a></li> </ul> <p>Mobile Hub-enabled features use Amazon Cognito for authentication and IAM for authorization. For more information, see <a href="#">User Sign-in</a> (p. 52).</p> <p>For more information, see <a href="#">Viewing AWS Resources Provisioned for this Feature</a> (p. 43).</p>
<p><b>Configuration options</b></p>	<p>This feature enables the following mobile backend capabilities:</p> <p><b>Messaging Service Integration</b></p> <ul style="list-style-type: none"> <li>• via <b>Firebase or Google Cloud Messaging (FCM/GCM)</b> (see <a href="#">Setting Up Android Push Notification</a> (p. 41))</li> <li>• via <b>Apple Push Notification service (APNs)</b> (see <a href="#">Setting Up iOS Push Notification</a> (p. 34))</li> </ul>

	For more information, see <a href="#">Configuring the Push Notifications Feature (p. 34)</a> .
<b>Quickstart demo features</b>	This feature adds the following to a quickstart app generated by Mobile Hub: <ul style="list-style-type: none"><li>• App is registered to receive and display push messages sent to the device individually, and those sent to a topic the device has been subscribed to.</li></ul>

## Configuring the Push Notifications Feature

When you include Push Notifications in your project, Mobile Hub creates an Amazon SNS Platform Application based on your choice of push platform. Mobile Hub also creates an Amazon Simple Notification Service topic named `ALL_DEVICES` and modifies the IAM role to allow your app to create a platform endpoint and subscribe that endpoint to the `ALL_DEVICES` topic and any others you configured when creating your project.

For more information on choosing a push provider, see [Setting Up Push Notification Services \(p. 34\)](#).

## Setting Up Push Notification Services

Using AWS Mobile Hub, you can enable a user push notification feature for your app. Push notification works with native platform support for sending push notifications, including Apple Push Notification service (APNs) for iOS apps and Firebase Cloud Messaging (FCM) for Android apps. Mobile Hub helps you configure push notifications for APNs or iOS; however you also must set up push notifications with the platforms you plan to use.

The topics in this section detail the setup you must complete with push notification support for iOS and Android apps and obtain data values Mobile Hub needs to configure your push notification feature.

Topics

- [Setting Up iOS Push Notification \(p. 34\)](#)
- [Setting Up Android Push Notification \(p. 41\)](#)

### Setting Up iOS Push Notification

Mobile Hub sends push notifications to iOS apps using Apple Push Notification service (APNs). To integrate this service with Mobile Hub, you must obtain and provide a certificate for APNs. To do this, you must prepare a certificate request, and then create an app ID and associated SSL certificate on the Apple Developer website. The certificate allows the Amazon Simple Notification Service server to send push notifications to the app identified by the App ID.

Topics

- [Generating a Certificate Request \(p. 34\)](#)
- [Setting up an App ID \(p. 35\)](#)
- [Configuring the App ID for Development Push Notifications \(p. 37\)](#)

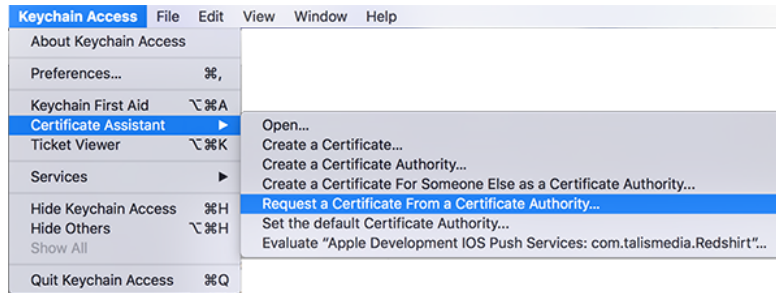
### Generating a Certificate Request

To create the certificate required by iOS to enable push notifications for your app, start by generating a certificate request on your Mac that you will use later to create the certificate.

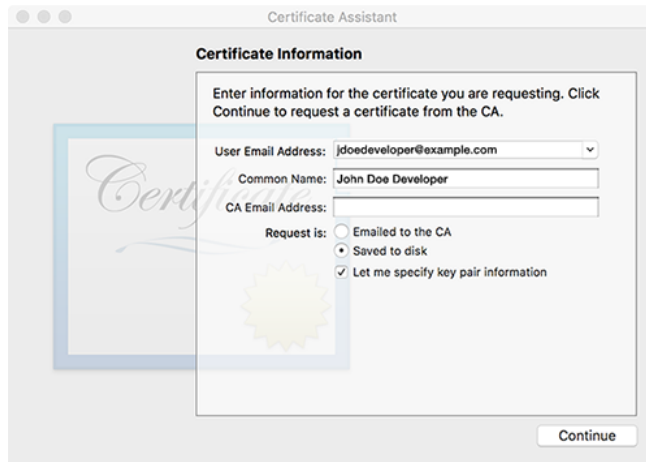


## To generate a certificate request

1. On your Mac, start the Keychain Access application.
2. From the **Keychain Access** menu, choose **Certificate Assistant** and then choose **Request a Certificate From a Certificate Authority...**



3. Enter your e-mail address and name.



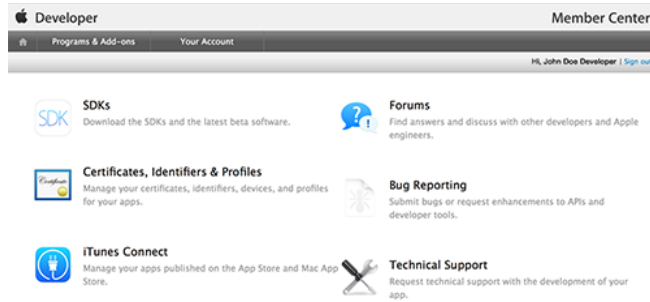
4. Select **Saved to disk** to create a file that contains the certificate request.

## Setting up an App ID

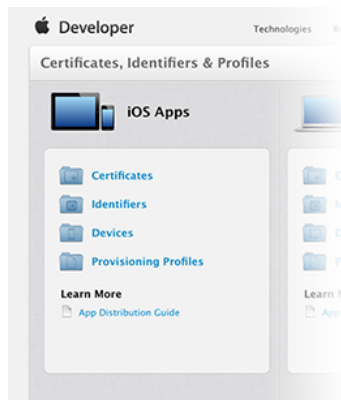
You need to provide an app ID for your app. Every app installed on a developer device needs an app ID. Typically, an app ID consists of a reversed web address, for example `com.amazon.mysampleapp`. You can use an existing app ID if it doesn't contain a wildcard character ("\*").

### To assign an app ID to your app

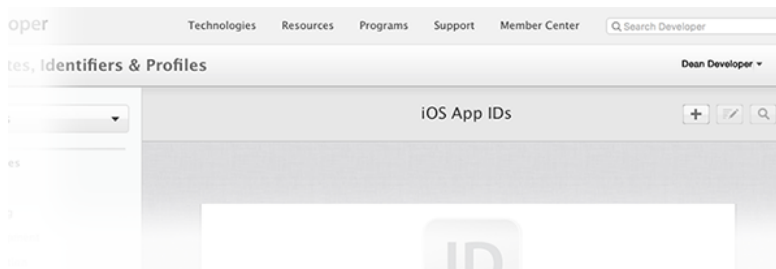
1. Sign into the Apple Developer Member Center website at <https://developer.apple.com/membercenter/index.action>.
2. Choose **Certificates, Identifiers & Profiles**.



3. In the **iOS Apps** section, choose **Identifiers**.



4. At the top of the list of your iOS apps IDs, select **+**.



5. Type a name for the new app ID.

## Registering an App ID

The App ID string contains two parts separated by a period (.)—an App ID Prefix that is defined as your Team ID by default and an App ID Suffix that is defined as a Bundle ID search string. Each part of an App ID has different and important uses for your app. [Learn More](#)

### App ID Description

Name:

You cannot use special characters such as @, &, \*, ', "

### App ID Prefix

Value:  (Team ID)

6. Choose the default selection for **App ID Prefix**.
7. For **App ID Suffix**, choose **Explicit App ID** and then enter the **Bundle ID** for your app. This ID must match the Bundle Identifier in your app's Info.plist.

App ID Suffix

**Explicit App ID**  
If you plan to incorporate app services such as Game Center, In-App Purchase, Data Protection, and iCloud, or want a provisioning profile unique to a single app, you must register an explicit App ID for your app.

To create an explicit App ID, enter a unique string in the Bundle ID field. This string should match the Bundle ID of your app.

Bundle ID:

We recommend using a reverse-domain name style string (i.e., com.domainname.appname). It cannot contain an asterisk (\*).

Wildcard App ID  
This allows you to use a single App ID to match multiple apps. To create a wildcard App

8. Under **App Services** choose **Push Notification**.

include CloudKit support (requires Xcode 6)

In-App Purchase

Inter-App Audio

Wallet

Push Notifications

VPN Configuration & Control

9. Choose **Continue**. Check that all values were entered correctly. The identifier must match your app's bundle identifier and app ID prefix.

Apple Pay:  Disabled

Wallet:  Disabled

Push Notifications:  **Configurable**

VPN Configuration & Control:  Disabled

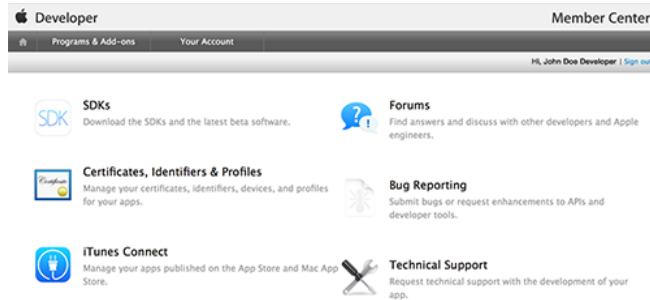
10. Choose **Submit** to register the new app ID.

## Configuring the App ID for Development Push Notifications

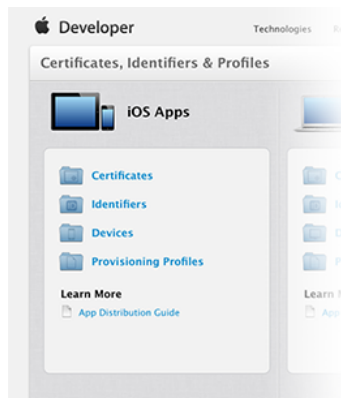
After creating a new app ID or choosing an existing explicit app ID, you must configure the app ID for push notifications.

### To configure an app ID for push notifications

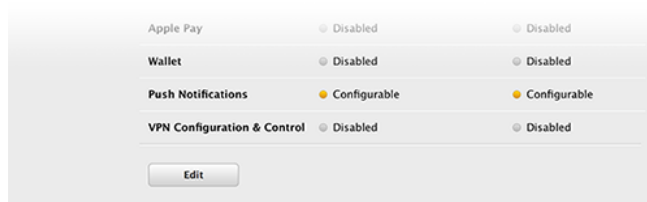
1. Sign into the Apple Developer Member Center website at <https://developer.apple.com/membercenter/index.action>.
2. Choose **Certificates, Identifiers & Profiles**.



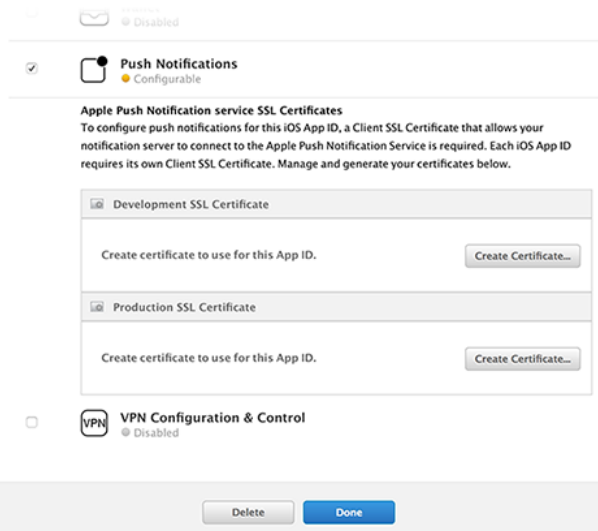
3. From the **iOS Apps** section, choose **Identifiers**.



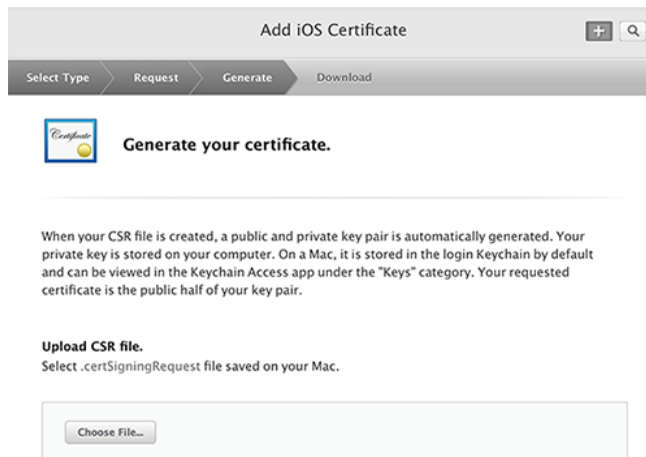
4. Select your newly created app ID from the list of iOS app IDs.
5. Choose **Edit**.



6. Under **Push Notifications** there are options to create a development SSL Certificate as well as a production SSL Certificate. Select **Create Certificate...** in the **Development SSL Certificate** section.



7. Choose **Continue** on the page that provides instructions on generating a Certificate Signing Request (CSR). This is the same certificate request created in [Generating a Certificate Request \(p. 34\)](#) and does not need to be created again.
8. In **Generate your certificate**, select **Choose File...** and then select the CSR file you created.

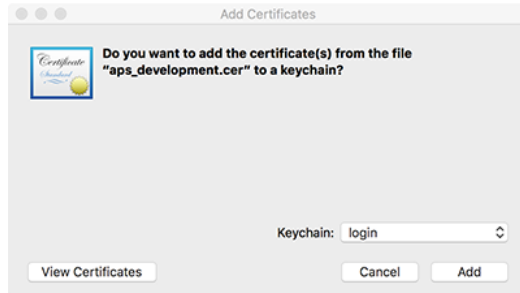


9. Choose **Generate**.
10. When the certificate is ready, choose **Download** and then save the certificate to your computer.

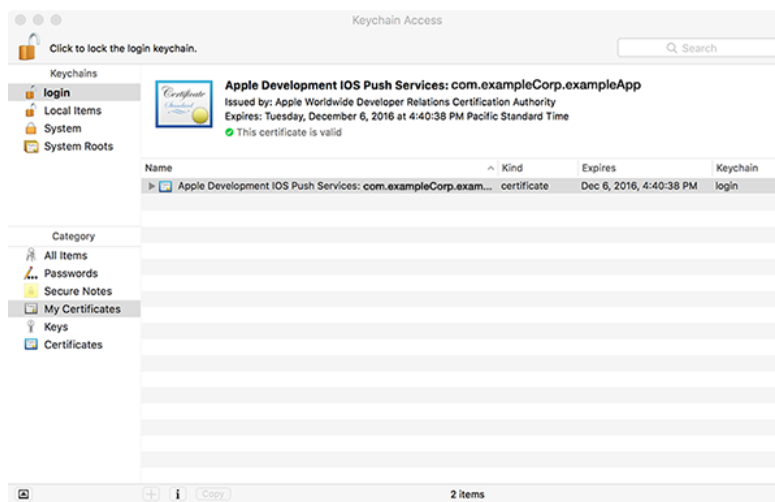


**Documentation**  
For more information on using and managing your certificates read:

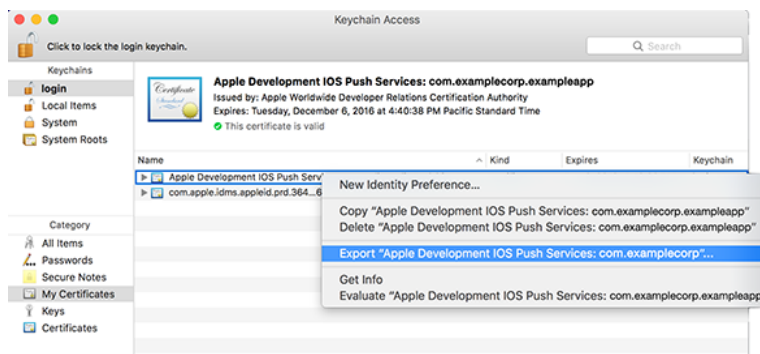
11. Double-click the downloaded certificate to install it to the Keychain on your Mac. In the **Add Certificates** dialog box, choose **Add**.



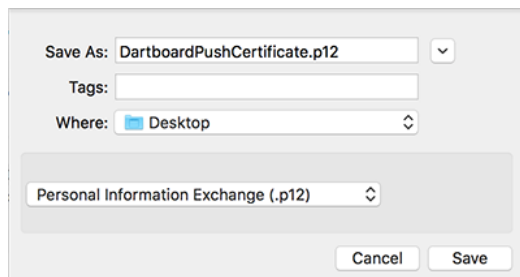
12. On your Mac, start the Keychain Access application.
13. In **My Certificates**, navigate to the certificate you just added. It should be called "Apple Development iOS Push Services:".



14. Context-select this certificate and then choose **Export...** from the context menu to export a file that contains the certificate.



15. Name the exported certificate "MobileHubPushCertificate.p12" and then save it to your computer. Do not provide an export password when prompted. You need to upload this certificate when creating your app in the Mobile Hub console.



Push notification is now enabled for your app in development mode. Prior to releasing your app on the App Store, you must repeat these steps but choose **Production Push SSL Certificate**.

## Setting Up Android Push Notification

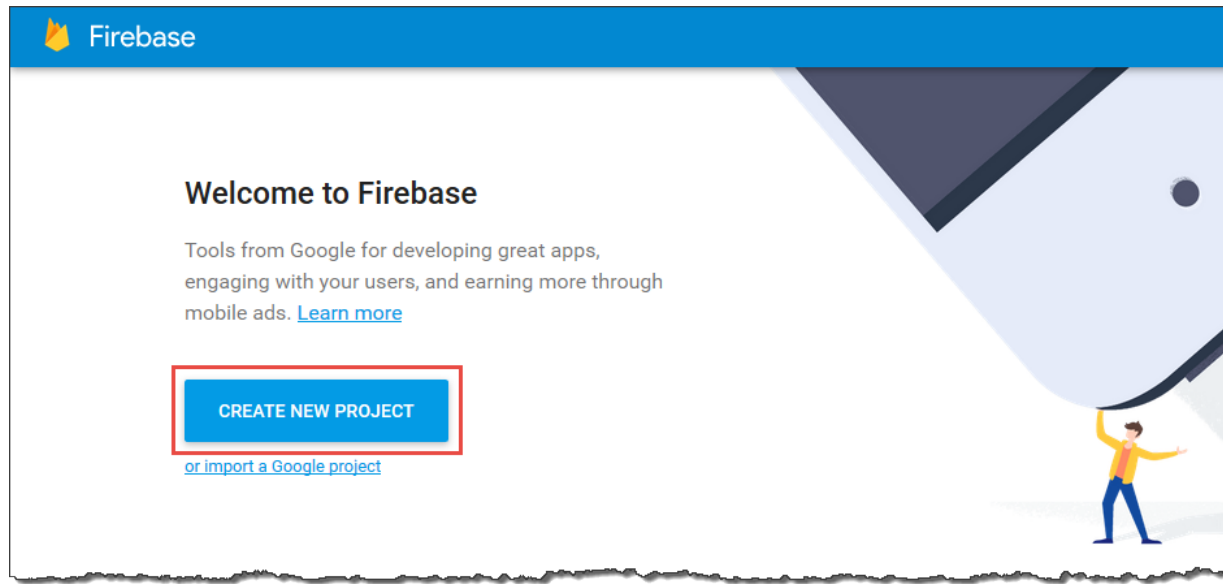
To send push notifications to Android apps, you must have credentials from either Firebase Cloud Messaging (FCM) or its predecessor, Google Cloud Messaging (GCM). This topic describes how to retrieve your credentials from FCM or GCM.

The credentials are: **server key** and a **sender ID** (also called project number). You get these credentials from a project, with cloud messaging enabled, that you have created either in the Firebase console or the Google Cloud Platform console.

Use FCM for new Android apps. Use GCM only if you have a preexisting GCM project that you have not yet updated for FCM support. To get your GCM app's credentials in this case, see [Obtain the Credentials of an Existing GCM App](#) (p. 43).

### To obtain your credentials from FCM

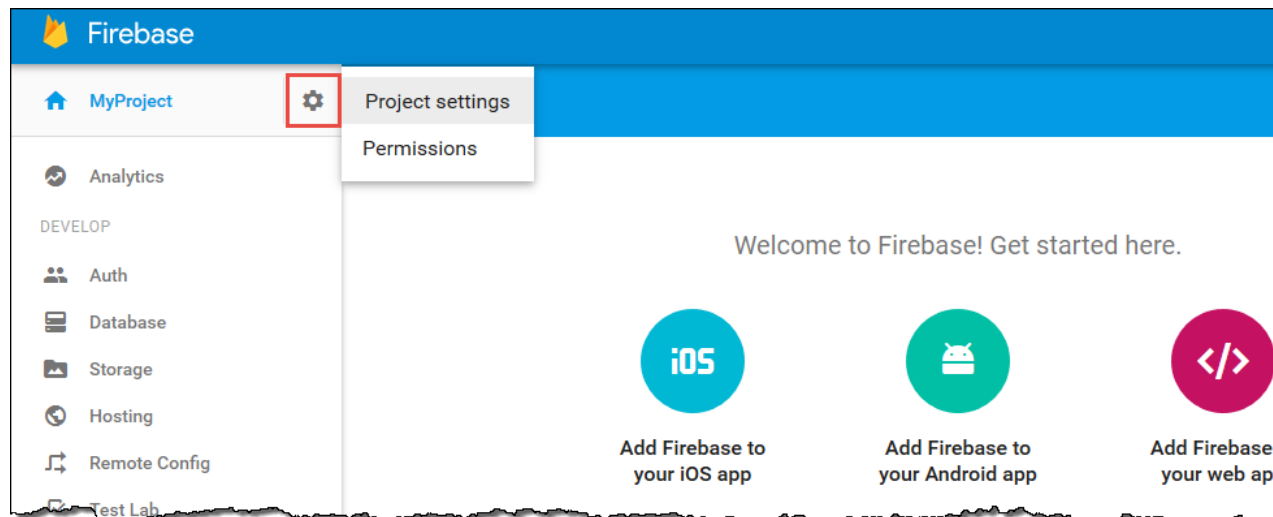
1. Go to the Firebase console at <https://console.firebase.google.com/>. If you are not signed in to Google, the link will take you to a sign-in page. After you sign in, you will see the Firebase console.
2. If you do not already have a Firebase project created that you want to use to enable Push Notifications:
  - a. Choose **Create New Project**.



- b. Type a project name, and then choose **Create Project**.

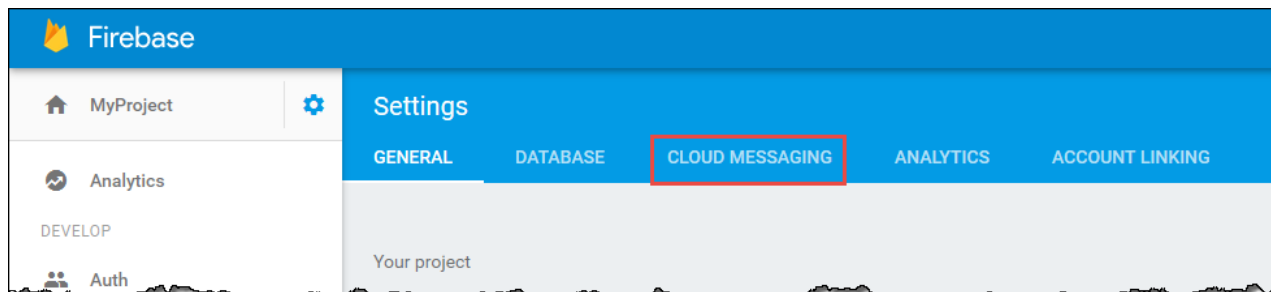
Firebase projects support push messaging by default.

3. In the left pane, to the right of your project name, choose the gear icon, and then choose **Project settings**.



4. In the top menu, choose **Cloud Messaging**.





5. Under **Project credentials**, you will find the server key and sender ID. Save these values somewhere you can access later.

## Obtain the Credentials of an Existing GCM App

### To obtain your credentials from GCM

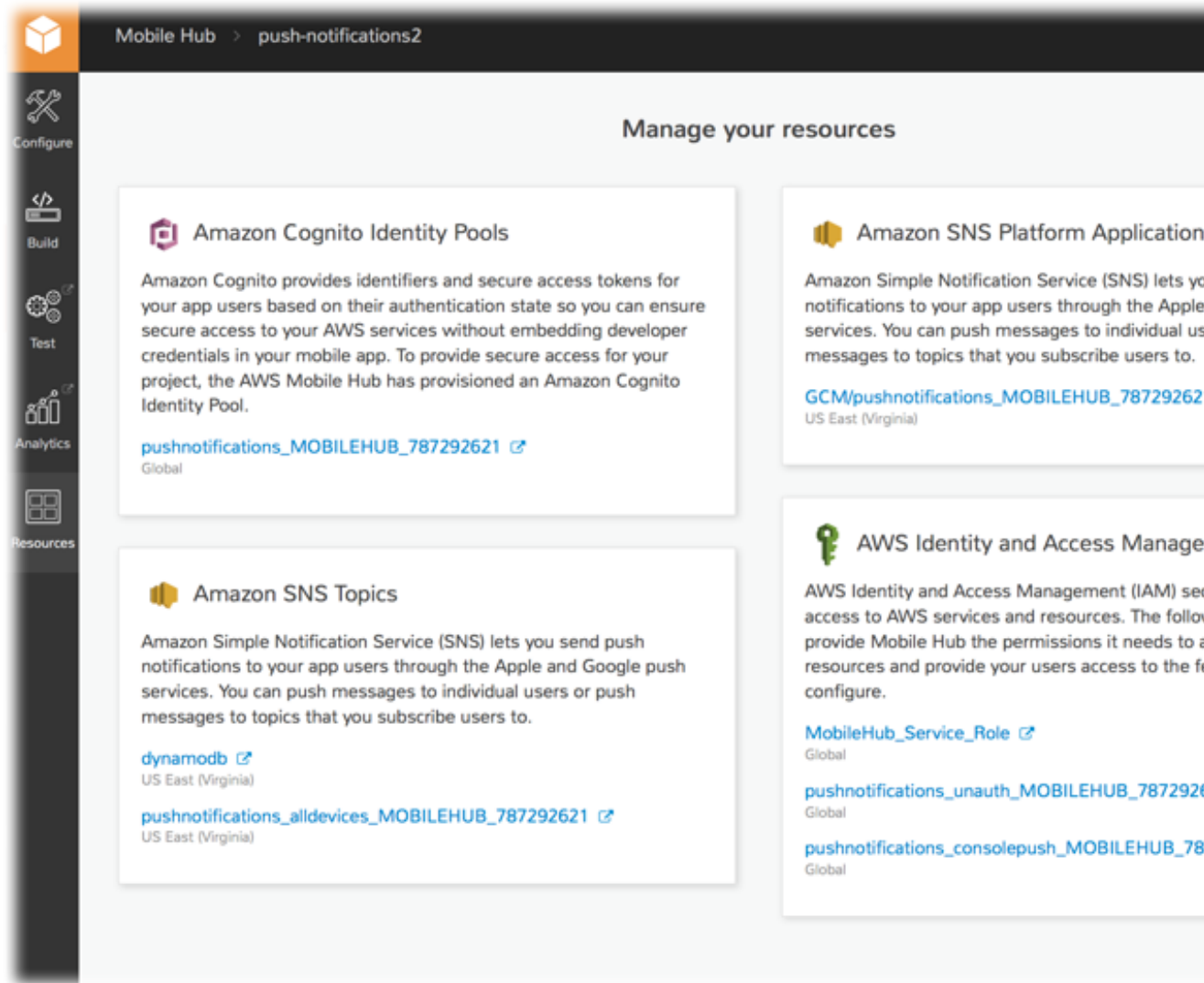
1. Go to the Google API Console at <https://console.developers.google.com>.
2. In the left pane, choose **Credentials**.
3. If you already have credentials for your app, your server key is shown in the **API keys** section. Save this key somewhere you can access later.
4. If you don't have credentials for your app, the console displays the **Credentials** dialog box. Create a server key by completing the following steps:
  - a. Choose **Create credentials**.
  - b. Choose **API key**.
  - c. Save the API key somewhere you can access later.
5. To retrieve your sender ID (also called project number), go to the Google Cloud Platform console at <https://console.cloud.google.com/>. Select your project from the **Project** menu. Then, choose the down arrow next to the project name.
6. Save the displayed project number somewhere you can access later.

#### Note

Project number is another name for sender ID.

## Viewing AWS Resources Provisioned for this Feature

The following image shows the Mobile Hub **Resources** pane displaying elements typically provisioned for the Push Notifications feature:



## Quickstart App Details

Upon launch, the Push Notification demo in a quickstart app automatically registers the installed app for push notifications with the configured provider. The app obtains a push token from the provider and passes it to create a new platform endpoint in the Amazon SNS application created for your project.

In addition to creating the Amazon SNS platform application, Mobile Hub provisions the **ALL\_DEVICES** topic that will be automatically subscribed to by the quickstart app. The IAM role is also modified to allow the quickstart app to create a platform endpoint and subscribe to the Amazon SNS topic. The demo also provides a list of all Amazon SNS topics that the device can subscribe to which includes the **ALL\_DEVICES** topic and any others you configured while creating your project. Selecting/unselecting a topic from the list of topics subscribes or unsubscribes the device's Amazon SNS platform endpoint from that topic.

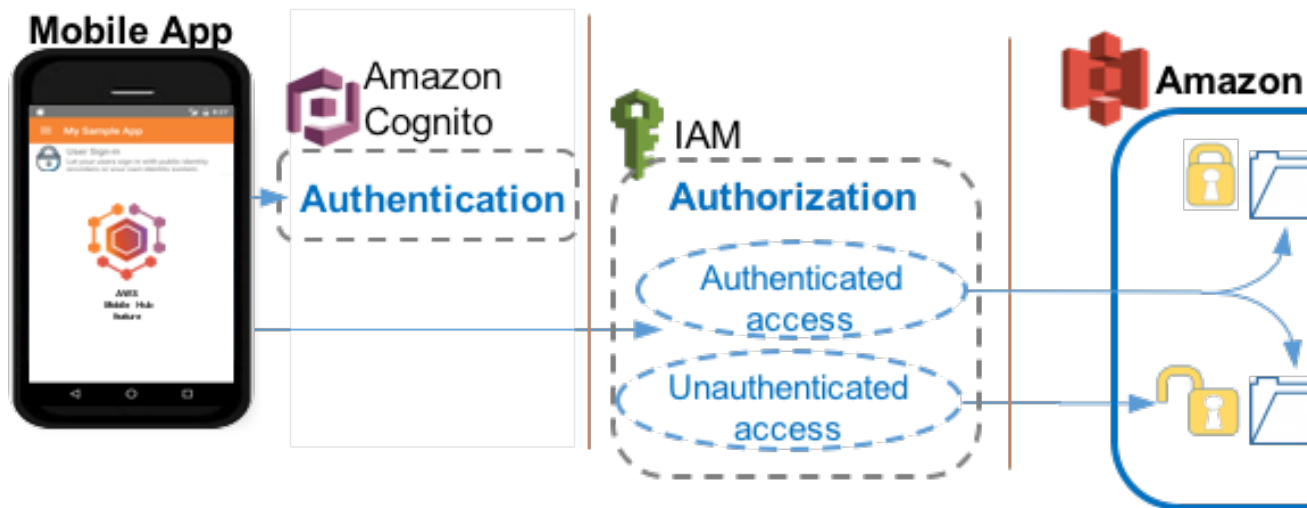
## Creating Test Push Notifications for the Quickstart app with Amazon SNS

To demonstrate the Push Notification feature and send messages to your quickstart app, you can open the [Amazon SNS console](#), choose the application, select the endpoint you wish to notify, and choose **Publish to Endpoint**. If you want to test publishing notifications via SNS topic, choose **Topics**, select the topic, and choose **Publish to topic**.

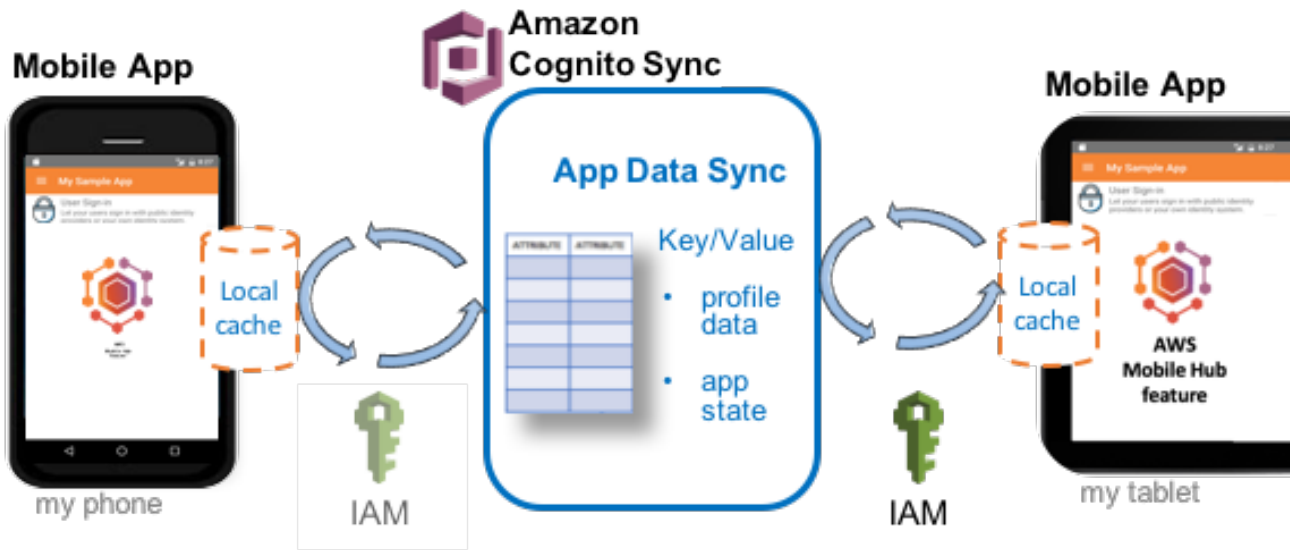
# User Data Storage

Choose AWS Mobile Hub User Data Storage to add cloud storage of user files, profile data, and app state to your mobile app. This feature enables sync and caching of data between devices using a simple programming model.

The following image shows access policy enforcement for public and private user files.



The following image shows user profile data sync for persisting user data and synchronizing it across devices.



The User Data Storage feature enables you to store user files such as photos or documents in the cloud, and it also allows you to save user profile data in key/value pairs, such as app settings or game state. When you select this feature, an [Amazon S3](#) bucket is created as the place your app will store user files.

Mobile Hub will also configure [Amazon Cognito Sync](#) so you can save user profile data in key/value pairs and synchronize that data across a user's authenticated devices.

Topics

- [User Data Storage At a Glance](#) (p. 47)
- [Viewing AWS Resources Provisioned for this Feature](#) (p. 48)
- [Quickstart App Details](#) (p. 49)

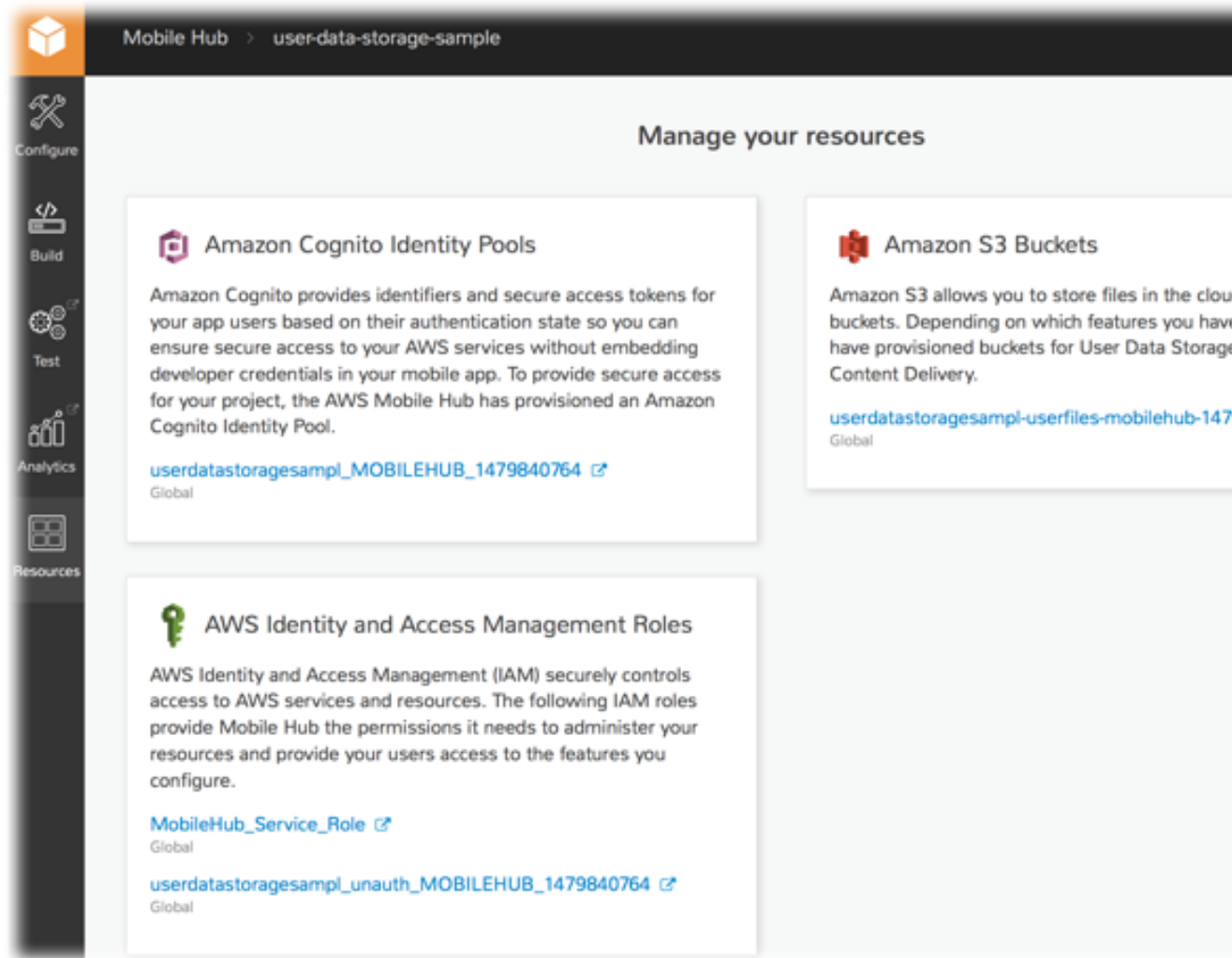
## User Data Storage At a Glance

<p><b>AWS services and resources configured</b></p>	<ul style="list-style-type: none"> <li>• <b>Amazon S3 bucket</b> (see <a href="#">Amazon Simple Storage Service Getting Started Guide</a>)  <a href="#">Concepts</a>   <a href="#">Console</a>   <a href="#">Pricing</a></li> <li>• <b>Amazon Cognito Sync</b> (see <a href="#">Amazon Cognito Sync</a>)  <a href="#">Concepts</a>   <a href="#">Console</a>   <a href="#">Pricing</a></li> </ul> <p>Mobile Hub-enabled features use Amazon Cognito for authentication and IAM for authorization. For more information, see <a href="#">User Sign-in</a> (p. 52).</p> <p>For more information, see <a href="#">Viewing AWS Resources Provisioned for this Feature</a> (p. 48).</p>
<p><b>Configuration options</b></p>	<p>This feature enables the following configuration options mobile backend capabilities:</p> <ul style="list-style-type: none"> <li>• Store user files and app data using Amazon S3.</li> </ul> <p>When you enable User Data Storage four folders are provisioned, each with a distinct access policy configuration:</p>

	<ul style="list-style-type: none"><li>• <code>private</code> - Each mobile app user can create, read, update, and delete their own files in this folder. No other app users can access this folder.</li><li>• <code>protected</code> - Each mobile app user can create, read, update, and delete their own files in this folder. In addition, any app user can read any other app user's files in this folder.</li><li>• <code>public</code> - Any app user can create, read, update, and delete files in this folder.</li><li>• <code>uploads</code> - Any app user can only create files in this folder.</li><li>• Synchronize data to the cloud and between a user's devices using Amazon Cognito Sync.</li></ul>
<b>Quickstart demo features</b>	<p>This feature adds the following to a quickstart app generated by Mobile Hub:</p> <ul style="list-style-type: none"><li>• File explorer for accessing sample files in the public and private folders in your S3 bucket.</li><li>• User settings sync persists user's choice of color theme to the cloud.</li></ul>

## Viewing AWS Resources Provisioned for this Feature

The following image shows the Mobile Hub **Resources** pane displaying elements typically provisioned for the User Data Storage feature.



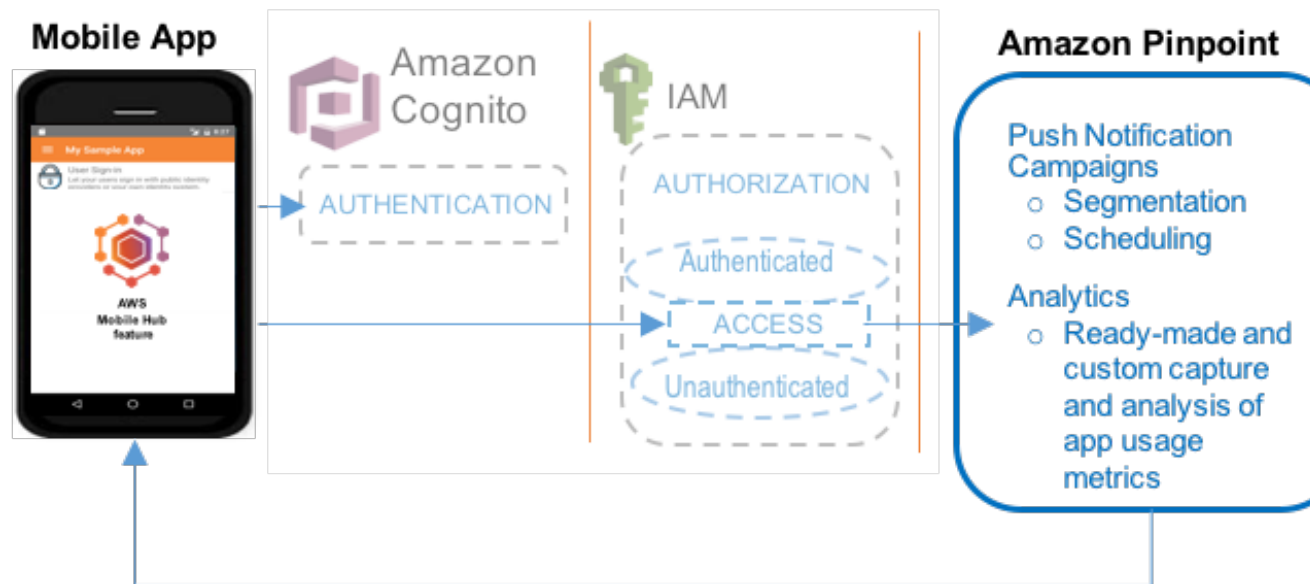
## Quickstart App Details

In the Mobile Hub quickstart app, the User Data Storage demo enables all users to see the contents of a public folder. When this feature is used in combination with User Sign-in, users who are signed in are able to access a private folder; unauthenticated users are not.

The demo also includes an option for the user to change the color scheme of the app. That choice is stored in Amazon Cognito Sync Profile. Any time the user returns to the app, their chosen theme is loaded from the stored user profile. If the user is authenticated, the same theme can sync across all devices they own.

# User Engagement

Choose the AWS Mobile Hub User Engagement feature to add push notification campaigns to reach your app users, and to understand how they are engaging with your app.



AWS Mobile Hub User Engagement helps you understand how your users use your app and enables you to engage them through push notification, based on a pattern you design.

User Engagement uses Amazon Pinpoint to carry out campaigns that interact with your app users based on the way you configure Amazon Pinpoint options. You decide which users receive notification, why a notification gets pushed to them, and the timing of when the notification is sent. You decide if the notification displays a text message, opens an application deep link, or passes a custom JSON statement to the client.

Amazon Pinpoint performs capture, visualization, and analysis of app usage event metrics that describe how your users use your app. You can choose Amazon Pinpoint default options or custom design the data that gets collected to align with your app design and campaign goals. Amazon Pinpoint enables you to use this data as a factor in your campaign parameters.



## User Engagement At a Glance

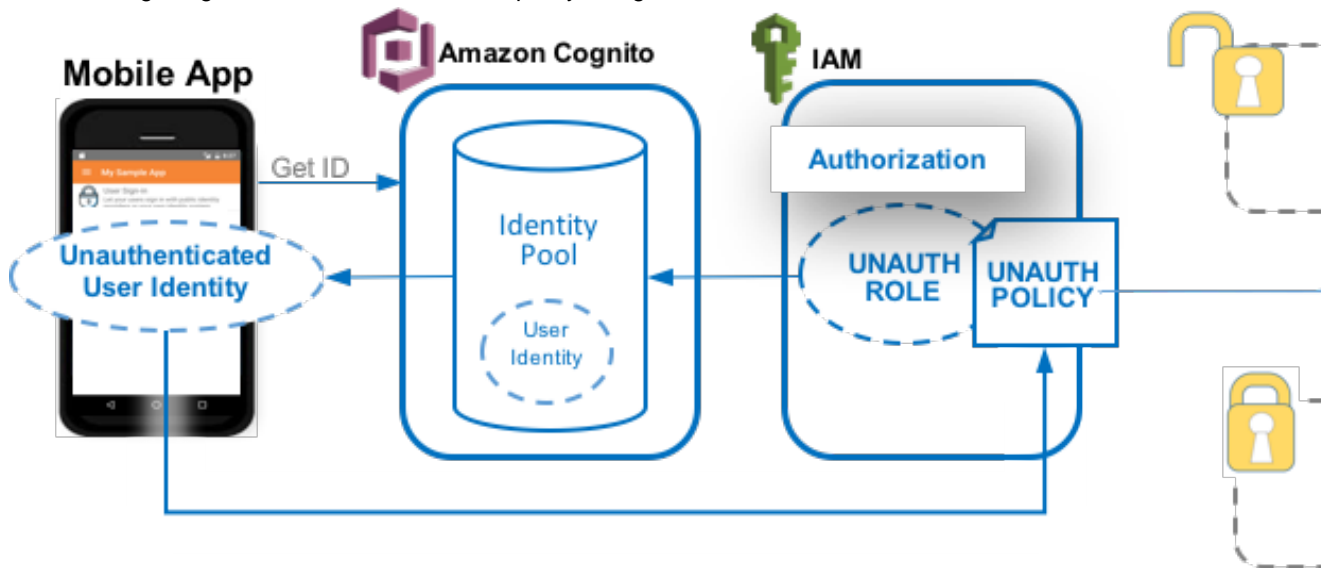
<b>AWS services and resources configured</b>	<ul style="list-style-type: none"><li>• <b>Amazon Pinpoint</b> (see <a href="#">Amazon Pinpoint Developer Guide</a>) <a href="#">Concepts</a>   <a href="#">Console</a></li></ul> <p>Mobile Hub-enabled features use Amazon Cognito for authentication and IAM for authorization. For more information, see <a href="#">User Sign-in (p. 52)</a>.</p>
<b>Configuration options</b>	<p>This feature enables the following mobile backend capabilities:</p> <ul style="list-style-type: none"><li>• Integrate Amazon Pinpoint campaigns into your mobile app.</li><li>• Integrate push notifications through APNs, GCM, and FCM.</li><li>• Integrate your app by downloading and reusing the code of the quickstart app, a package of native iOS and Android SDKs, plus helper code and developer guidance, all of which are dynamically generated to match your Mobile Hub project.</li></ul>
<b>Quickstart app demos</b>	<p>This feature adds the following functionality to a quickstart app generated by Mobile Hub:</p> <ul style="list-style-type: none"><li>• Demonstrate enabling the app user to receive campaign notifications.</li><li>• Demonstrate providing the app user with a view of an Amazon Pinpoint data visualization, on their mobile phone.</li></ul>

# User Sign-in

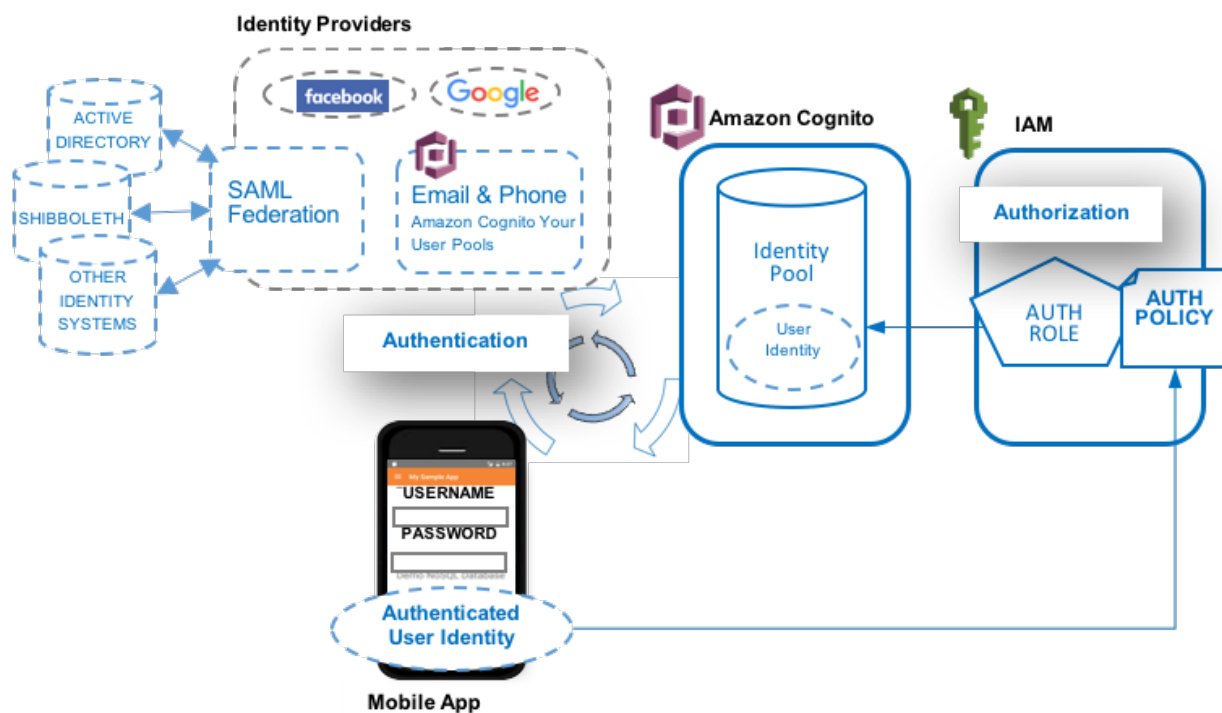
Choose the AWS Mobile Hub User Sign-in mobile backend feature to add AWS user authentication and identity access management to your mobile app. Your app can include access to AWS resources for unauthenticated users and/or integrate the sign-in process for one or more identity providers like Facebook, Google or your own user directory created in AWS.

You can also federate your existing user directory outside of AWS that uses an identity provider like Microsoft Active Directory Federation Services or Shibboleth. This allows your users to sign-in to use your backend features in AWS using their familiar credentials.

The following image shows a resource access policy being enforced for an unauthenticated user.



The following image shows a resource access policy being enforced for an authenticated user.



This feature enables you to configure how your users gain access to AWS resources and services used by your app, either with no sign in process or through authentication provided by one or more identity providers. In both cases, AWS identity creation and credentials are provided by [Amazon Cognito Identity](#), and access authorization comes through [AWS Identity and Access Management \(IAM\)](#).

When you create a project, Mobile Hub provisions the AWS identity, user role, and access policy configuration required to allow all users access to unrestricted resources. When you add the User Sign-in feature to your app, you are able to restrict access to allow only those who sign in with credentials validated by an identity provider to use protected resources. Through Amazon Cognito Identity, your app user obtains AWS credentials to directly access the AWS services that you enabled and configured for your Mobile Hub project. Both authenticated and unauthenticated users are granted temporary, limited-privilege credentials with the same level of security enforcement.

Amazon Cognito can federate validated user identities from multiple identity providers to a single AWS identity. Mobile Hub helps you integrate identity providers into your mobile app so that users can sign in using their existing credentials from Facebook, Google, and your own identity system. You can also create and configure your own email- and password-based user directory using Amazon Cognito Your User Pools.

#### Topics

- [User Sign-in Feature At a Glance \(p. 54\)](#)
- [Configuring User Sign-in \(p. 54\)](#)
- [Setting Up User Authentication \(p. 56\)](#)
- [Viewing AWS Resources Provisioned for this Feature \(p. 72\)](#)
- [Quickstart App Details \(p. 73\)](#)

## User Sign-in Feature At a Glance

<b>AWS services and resources configured</b>	<ul style="list-style-type: none"><li>• <b>Amazon Cognito</b> <a href="#">Concepts</a>   <a href="#">Console</a>   <a href="#">Pricing</a></li><li>• <b>Amazon Cognito Identity Pool</b> (see <a href="#">Using Federated Identities</a>)</li><li>• <b>Amazon Cognito Your User Pools</b> (see <a href="#">Creating and Managing User Pools</a>)</li><li>• <b>Amazon Cognito SAML Federation</b> (see <a href="#">Overview of Configuring SAML 2.0-Based Federation</a>)</li><li>• <b>IAM role and security policies</b> (see <a href="#">Controlling Access to Mobile Hub Projects (p. 117)</a>) <a href="#">Concepts</a>   <a href="#">Console</a>   <a href="#">Pricing</a></li></ul> <p>For more information, see <a href="#">Viewing AWS Resources Provisioned for this Feature (p. 72)</a>.</p>
<b>Configuration options</b>	<p>This feature enables the following mobile backend capabilities:</p> <p><b>Sign-in Providers</b> (users gain greater access when they sign in)</p> <ul style="list-style-type: none"><li>• via <b>Google</b> authentication (see <a href="#">Setting Up Google Authentication (p. 60)</a>)</li><li>• via <b>Facebook</b> authentication (see <a href="#">Setting Up Facebook Authentication (p. 57)</a>)</li><li>• via <b>Email and Password</b> authentication (see <a href="#">User Sign-in Providers (p. 55)</a>)</li><li>• via <b>SAML Federation</b> authentication (see <a href="#">User Sign-in Providers (p. 55)</a>)</li></ul> <p><b>Required Sign-in</b> (authenticated access)</p> <p><b>Optional Sign-in</b> (users gain greater access when they sign in)</p> <p>For more information, see <a href="#">Configuring User Sign-in (p. 54)</a></p>
<b>Quickstart demo features</b>	<p>This feature adds the following to a quickstart app generated by Mobile Hub:</p> <ul style="list-style-type: none"><li>• Unauthenticated access (if allowed by your app's configuration), displaying the ID that AWS assigns to the app instance's device.</li><li>• Sign-in screen that authenticates users using the selected method: Facebook, Google, or Custom.</li><li>• With <b>Optional Sign-in</b> and <b>Require Sign-in</b>, the app demonstrates an access barrier to protected folders for unauthenticated users.</li></ul>

## Configuring User Sign-in

The following options are available for configuring your users' sign-in experience.

## User Sign-in Providers

### Facebook

To enable Facebook user authentication, register your application with Facebook.

If you already have a registered Facebook app, copy the **App ID** from the Facebook Developers **App Dashboard**. Paste the ID into the Facebook **App ID** field and choose **Save Changes**.

If you do not have a Facebook App ID yet, you'll need to create one before you can integrate Facebook in your mobile app. The Facebook Developers portal takes you through the process of setting up your Facebook application.

For full instructions on integrating your application with Facebook, see [Setting Up Facebook Authentication \(p. 57\)](#).

### Google

To authenticate your users through Google, fully integrate your sample app with Google+ Sign-in.

If you already have a registered Google Console project with the Google+ API, a web application OAuthClient and a client ID for the platform of your choice set up, then copy and paste the Google Web App Client ID and client ID(s) from the Google Developers Console into those fields and choose **Save Changes**.

Regardless of the platform you choose (Android or iOS), you'll need to at least create the following.

- A Google Console project with the Google+ API enabled (used for Google Sign-in)
- A web application OAuth client ID
- An iOS and/or Android client ID, depending on which platform you are supporting

For full instructions on integrating your application with Google+, see [Setting Up Google Authentication \(p. 60\)](#).

### Email and Password

Choose Email and Password sign-in when you want to create your own AWS-managed user directory and sign-in process for your app's users. Configure the characteristics of their sign-in experience by:

- Selecting user login options (*email, username, and/or phone number*)
- Enabling multi-factor authentication (*none, required, optional*) which adds delivery of an entry code via text message to a user's phone, and a prompt to enter that code along with the other factor to sign-in
- Selecting password character requirements (*minimum length, upper/lower cases, numbers or special characters allowed*).

### SAML Federation

SAML Federation enables users with credentials in your existing identity store to sign in to your mobile app using their familiar username and password. A user signs into to your identity provider (IdP) which is configured to return a validating SAML assertion. Your app then uses Amazon Cognito Federated Identities to exchange the SAML assertion for typical temporary, limited privilege credentials to access your AWS backend services.

SAML 2.0 (Security Assertion Markup Language 2.0) is an open standard used by many IdPs, including Microsoft Active Directory Federation Service and Shibboleth. Your IdP must be SAML 2.0 compatible to use this Mobile Hub option. To establish federation between AWS and your IdP the two systems must exchange SAML federation metadata. AWS federation metadata can be found at <https://signin.aws.amazon.com/static/saml-metadata.xml>. This xml file demonstrates the form that your IdP's metadata should take. For more information on SAML federation metadata for your IdP, see [Integrating Third-Party SAML Solution Providers with AWS](#).

To implement this exchange, view your IdP's documentation to understand how to use the AWS federation metadata file to register AWS as a service provider. Then provide upload your IdP's federation metadata file using SAML Federation page of the Mobile Hub console.

To learn more about how AWS supports SAML federation, see [Overview of Configuring SAML 2.0-Based Federation](#).

## User Sign-in Requirement

### Sign-in is optional

Users have the option to sign in (authenticate) with your chosen sign-in identity provider(s) or users can skip sign-in (unauthenticated). Your app receives temporary, limited privilege access credentials from Amazon Cognito Identity as either an authenticated user or an unauthenticated guest user so that your app can access your AWS services securely.

### Sign-in is required

Users are required to sign in with one of your chosen sign-in providers. Your app receives temporary, limited privilege access credentials from Amazon Cognito Identity as an authenticated user so that your app can access your AWS services securely.

## User Sign-in and AWS Identity and Access Management (IAM)

When your mobile app is saved, Mobile Hub creates an Amazon Cognito identity pool and a new IAM role. These are used to generate temporary AWS credentials for the quickstart app users to access your AWS resources. The AWS IAM role security policies are updated based on the sign-in features enabled.

At this point, your mobile project is set up for users to sign in. Each chosen identity provider has been added to the login screen of the quickstart app.

For more information, see [Using AWS Managed Policies to Control Access to Mobile Hub Projects \(p. 117\)](#).

## Setting Up User Authentication

With AWS Mobile Hub, you can enable a user sign-in feature for your app. User sign-in works with various user authentication services, including Facebook, Google, and custom authentication. Mobile Hub helps you to configure sign-in with Facebook, Google, or your own identity system; however, you may also need to set up user authentication with the different authentication services you plan to use.

Learn more about how Amazon Cognito performs authentication using external identity providers, see [Understanding Amazon Cognito Authentication](#).

The topics in this section detail the setup you must complete with various user authentication services and how to obtain the data values Mobile Hub needs to configure your sign-in feature.

### Topics

- [Setting Up Facebook Authentication \(p. 57\)](#)
- [Setting Up Google Authentication \(p. 60\)](#)
- [Setting Up Custom Authentication \(p. 72\)](#)

## Setting Up Facebook Authentication

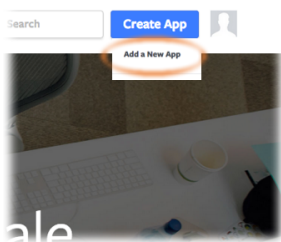
You must first register your application with Facebook by using the [Facebook Developers portal](#).

Mobile Hub generates code that enables you to use Facebook to provide federated authentication for your mobile app users. This topic explains how to set up Facebook as an identity provider for your app.

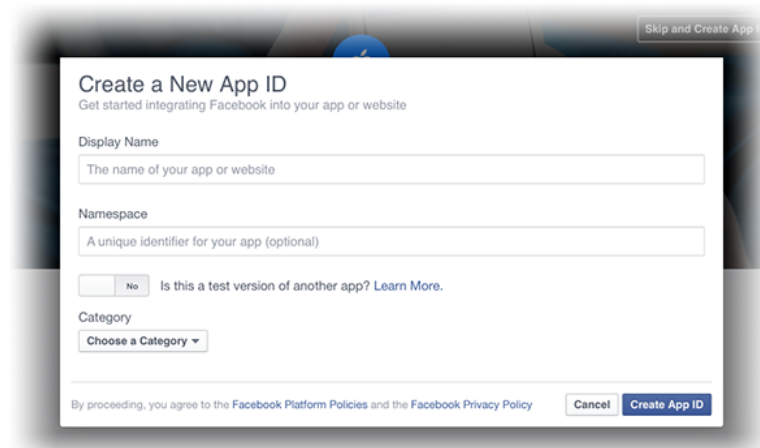
If you already have a Facebook app ID, copy and paste it into the **Facebook App ID** field in the Mobile Hub console, and choose **Save changes**.

### To get a Facebook app ID

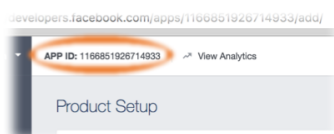
1. In the [Facebook Developers portal](#), sign in with your Facebook credentials.
2. From **Created App**, choose **Add a New App** (note: this menu label will be **My Apps** if you have previously created an app).



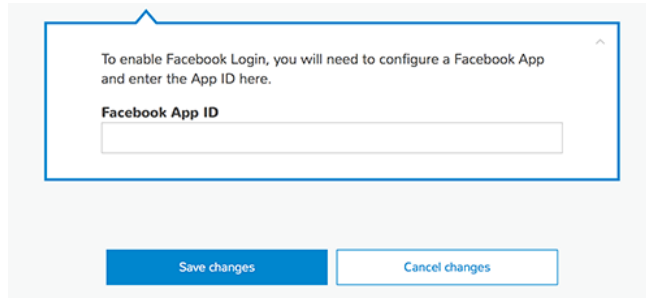
3. If asked, choose the platform of your app that will use Facebook sign-in, and **basic setup**.
4. Type a display name for your app, select a category for your app from the **Category** drop-down list, and then choose **Create App ID**.



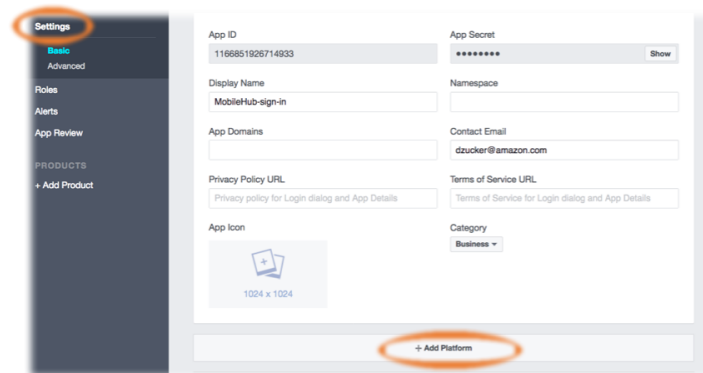
5. Complete the **Security Check** that appears. Your new app then appears in the **Dashboard**.



6. Copy the App ID and paste it into the **Facebook App ID** field in the Mobile Hub console.



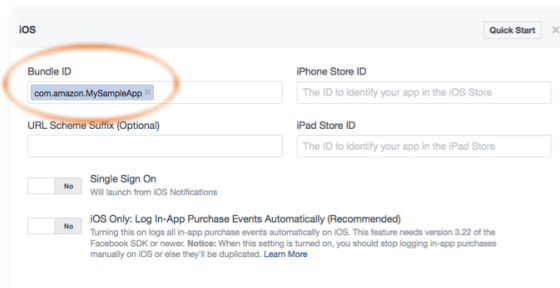
7. In the Facebook Developer portal's left hand navigation list, choose **Settings**, then choose **+ Add Platform**.



8. Choose your platform and provide information about your Mobile Hub app that Facebook will use for integration during credential validation.

#### For iOS:

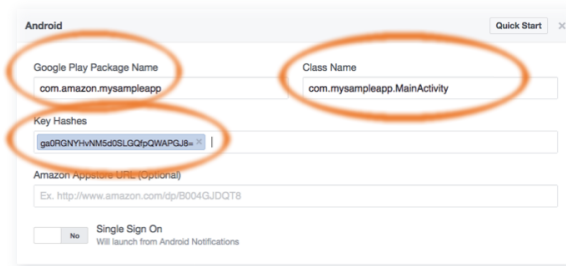
- Add your app's **Bundle ID**. (ie. `com.amazon.YourProjectName`). To use the AWS Mobile Hub sample app project, set your this value to `com.amazon.MySampleApp`.



#### For Android:

- a. Provide your app's **Google Play Package Name**. (ie. `com.yourprojectname`). To use the AWS Mobile Hub sample app project, set this value to `com.amazon.mysampleapp`.
- b. Provide your **Class Name** that handles deep links (ie. `com.yourprojectname.MainActivity`). To use the AWS Mobile Hub sample app project, set your class name to `com.mysampleapp.MainActivity`.





- c. Provide your app's Facebook development **Key Hashes**. This is a value that you generate via a terminal in your development environment, and is unique to that environment.

To generate a development key for your Android environment on Mac, run the following command line.

```
keytool -exportcert -alias androiddebugkey -keystore ~/.android/  
debug.keystore | openssl sha1 -binary | openssl base64
```

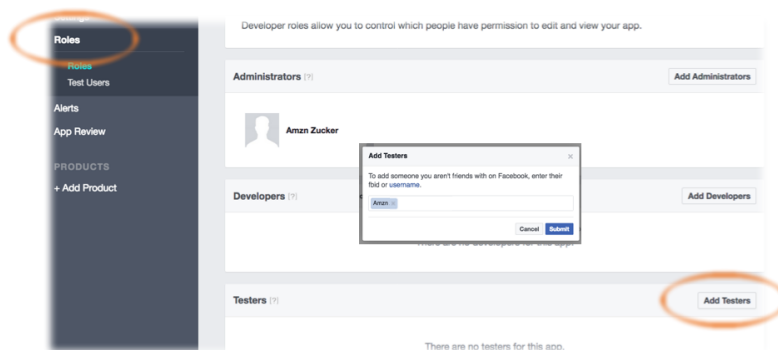
To generate a development key for your Android environment on Windows, run the following command line.

```
keytool -exportcert -alias androiddebugkey -keystore %HOMEPATH%  
\.android\debug.keystore | openssl sha1 -binary | openssl base64
```

For more information, choose the **Quick Start** button in the upper left of the Facebook Developer Portal Add Platform dialog.

9. In the Facebook Developers portal, choose **Save changes**, then **Use this package name** if a dialog appears saying that Google Play has an issue with your package name.
10. Only users with roles assigned in the Facebook portal will be able to authenticate through your app while it is in development (not yet published).

To authorize users, in the Facebook Developer portal's left hand navigation list, choose **Roles**, then **Add Testers**. Provide a valid Facebook ID.



11. In the Mobile Hub console, choose **Save changes**.

For more information about integrating with Facebook Login, see the [Facebook Getting Started Guide](#).

## Setting Up Google Authentication

With AWS Mobile Hub, you can configure a working Google Sign-In feature for both Android and iOS apps. To fully integrate Google Sign-In with your sample app, Mobile Hub needs information that you must first obtain through Google's setup process. The process has several parts, one of which is required regardless of which platforms you're supporting with your app. There are other parts to complete only for specific platforms:

- Create a Google Developers project and an OAuth Web Application Client ID (required for **all apps** regardless of platform)
- Create an OAuth Android client ID (required for all **Android apps**)
- Create an OAuth iOS client ID (required for all **iOS apps**)

This section details the Google Sign-In requirements as well as how to integrate Google Sign-In for both iOS and Android apps.

### Topics

- [Creating a Google Developers Project and OAuth Web Client ID \(p. 60\)](#)
- [Creating an OAuth Android Client ID \(p. 65\)](#)
- [Creating an OAuth iOS Client ID \(p. 68\)](#)
- [Verifying All Platform Client IDs \(p. 71\)](#)

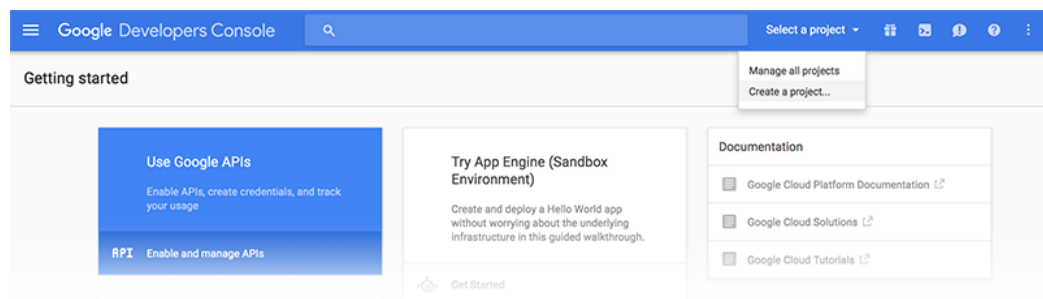
## Creating a Google Developers Project and OAuth Web Client ID

Before you enable Google Sign-In in an app, you must create a project in the Google Developers Console. Google recommends using a single project to create and manage all of the platform instances of your app, such as iOS, Android, and web.

Each platform requires its own OAuth client ID, which you obtain through the project you create for your app in the Google Developers Console. The first thing you must do is create a project for your app in the Google Developers Console that has the Google+ API enabled, and then enable an OAuth web client ID that Amazon Cognito uses to enable user authentication for your app.

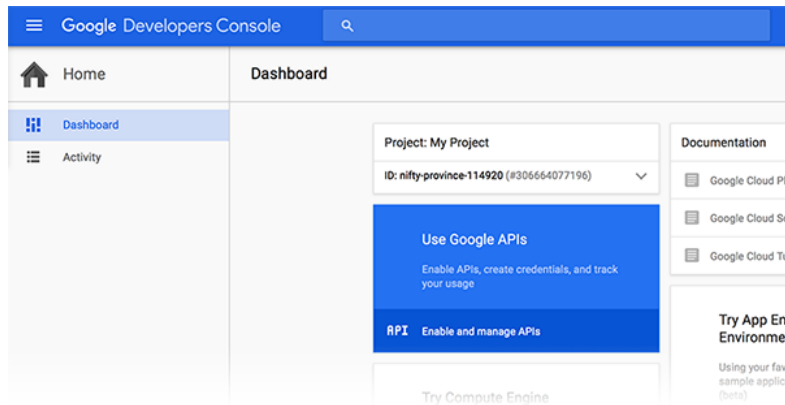
### To create a Google Developers project and OAuth web client ID

1. Go to the Google Developers Console at <https://console.developers.google.com>.
2. If you have not created a project yet, choose **Select a project** from the menu bar, and then choose **Create a project...**

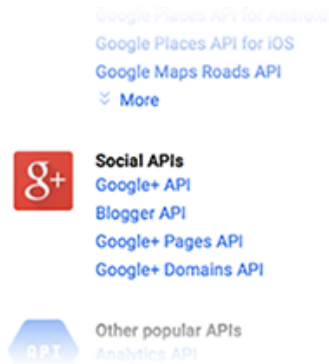


3. Complete the form that is displayed to create your new project.

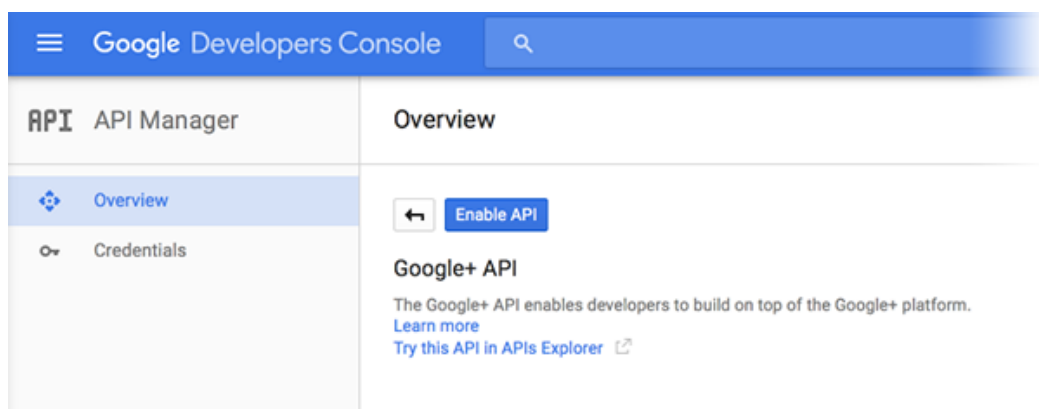
4. In the **Dashboard** for your project, go to the **Use Google APIs** section and then choose **Enable and manage APIs**.



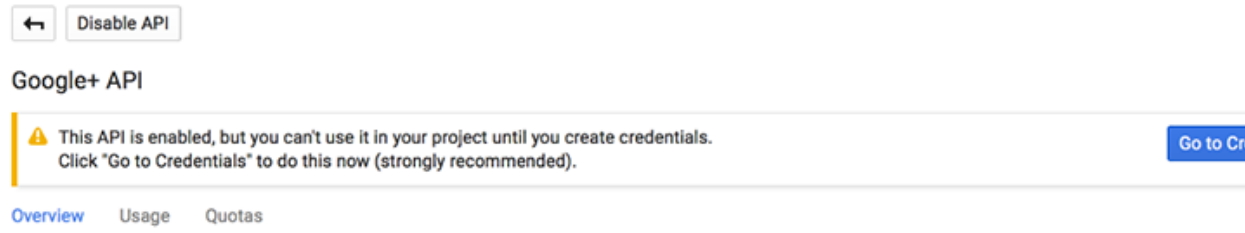
5. In the API Manager, in the **Social APIs** section, choose **Google+ API**.



6. In the **Overview** for Google+ API, choose **Enable API**.



7. A message appears to inform you that the API is enabled but that it requires credentials before you can use it. Choose **Go to Credentials**.



- Your Mobile Hub sample app authenticates users through Amazon Cognito Identity, so you need an OAuth web application client ID for Amazon Cognito. In **Credentials**, choose **client ID** from the links in the first step.

## Credentials

### Add credentials to your project

#### 1 Find out what kind of credentials you need

We'll help you set up the correct credentials

If you wish you can skip this step and create an [API key, client ID](#), or [service account](#)

#### Which API are you using?

Determines what kind of credentials you need.

Google+ API

#### Where will you be calling the API from?

Determines which settings you'll need to configure.

- A message appears to inform you that you must set a product name. Choose **Configure consent screen**.

## Credentials



### Create client ID

To create an OAuth client ID, you must first set a product name on the consent screen

Configure consent screen

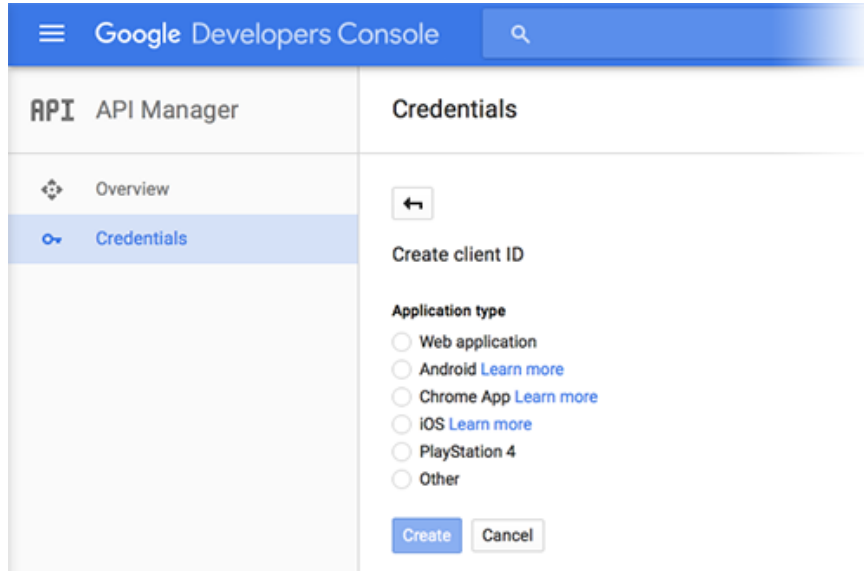
#### Application type

- Web application
- Android [Learn more](#)
- Chrome App [Learn more](#)
- iOS [Learn more](#)
- PlayStation 4
- Other

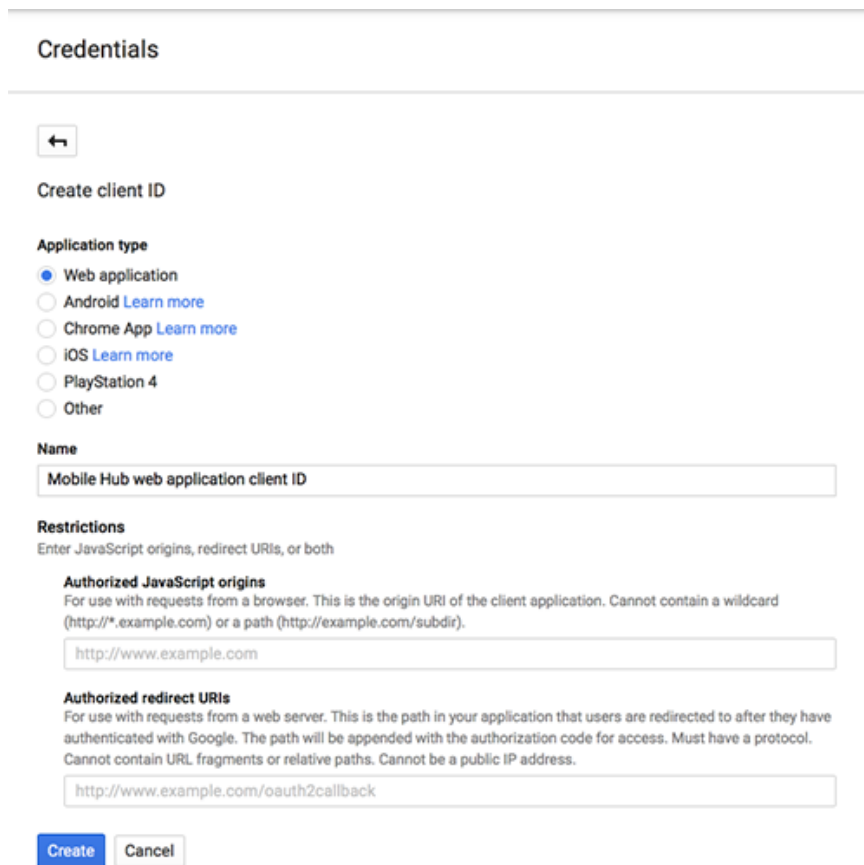
- In **OAuth consent screen**, enter the name of your app in **Product name shown to users**. Leave the remaining fields blank. Then choose **Save**.

The screenshot shows the AWS IAM console interface. At the top, there is a search bar and the word 'console'. Below that, the page title is 'Credentials'. There are three tabs: 'Credentials', 'OAuth consent screen' (which is selected), and 'Domain verification'. The main content area contains several form fields: 'Email address' with a dropdown menu showing '@gmail.com'; 'Product name shown to users' with a text input field containing 'Mobile Hub Sample App'; 'Homepage URL (Optional)' with an empty text input field; 'Product logo URL (Optional)' with a text input field containing 'http://www.example.com/logo.png'; a logo preview box with the text 'This is how your logo will look to end users' and 'Max size: 120x120 px'; 'Privacy policy URL (Optional)' with an empty text input field; and 'Terms of service URL (Optional)' with an empty text input field. At the bottom of the form are 'Save' and 'Cancel' buttons. On the right side, there is a help icon and a text block that reads: 'The consent screen is shown to users when they log in to their private account. It will be used for all applications in the project. You must provide a privacy policy and product logo to work.'

11. In **Create client ID**, choose **Web application**.



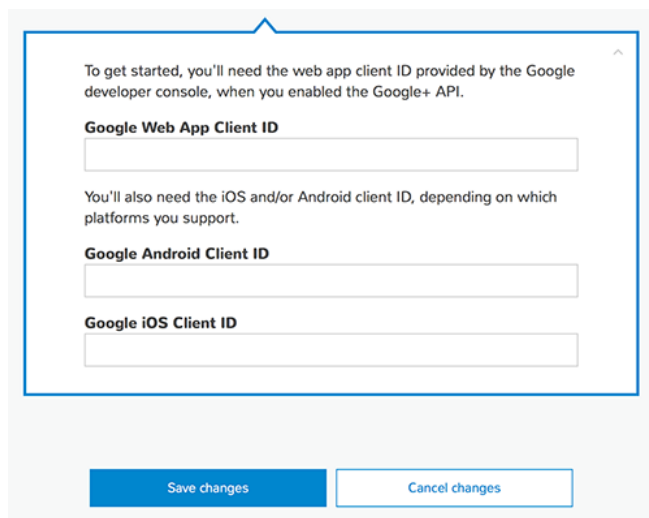
12. In **Name**, enter a name for the web client credentials for your app. Leave the **Authorized JavaScript origins** and **Authorized Redirect URIs** fields blank. Mobile Hub configures this information indirectly through Amazon Cognito Identity integration. Choose **Create**.



13. In the **OAuth client** pop-up, copy and save the value that was generated for your client ID. You will need the client ID to implement Google Sign-In in your Mobile Hub app. After you copy the client ID, choose **OK**.



14. Paste the web application client ID value into the Mobile Hub **Google Web App Client ID** field for your project.



## Creating an OAuth Android Client ID

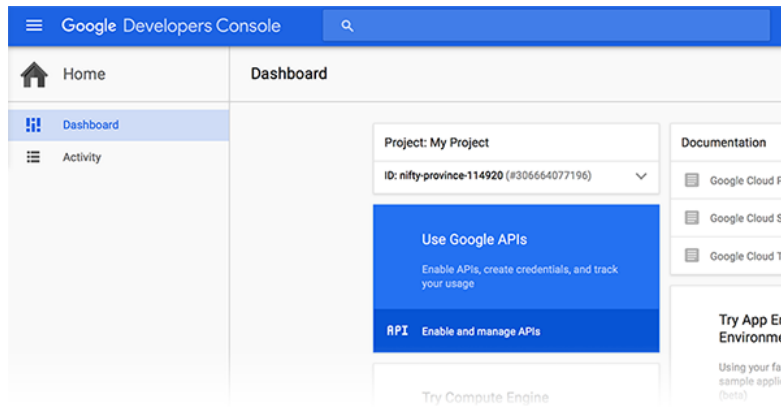
To enable Google Sign-In for your Android Mobile Hub sample app, you must create an Android OAuth client ID so that the Android sample app that is generated by Mobile Hub can access Google APIs directly and manage token lifecycle through Amazon Cognito Identity. This Android OAuth client ID is in addition to the Web application OAuth client ID you created while [Creating a Google Developers Project and OAuth Web Client ID](#) (p. 60).

To integrate Google Sign-In for your Android sample app, you must generate an Android OAuth client ID in the Google Developers Console. You will provide this client ID to Mobile Hub during the Google Sign-In configuration.

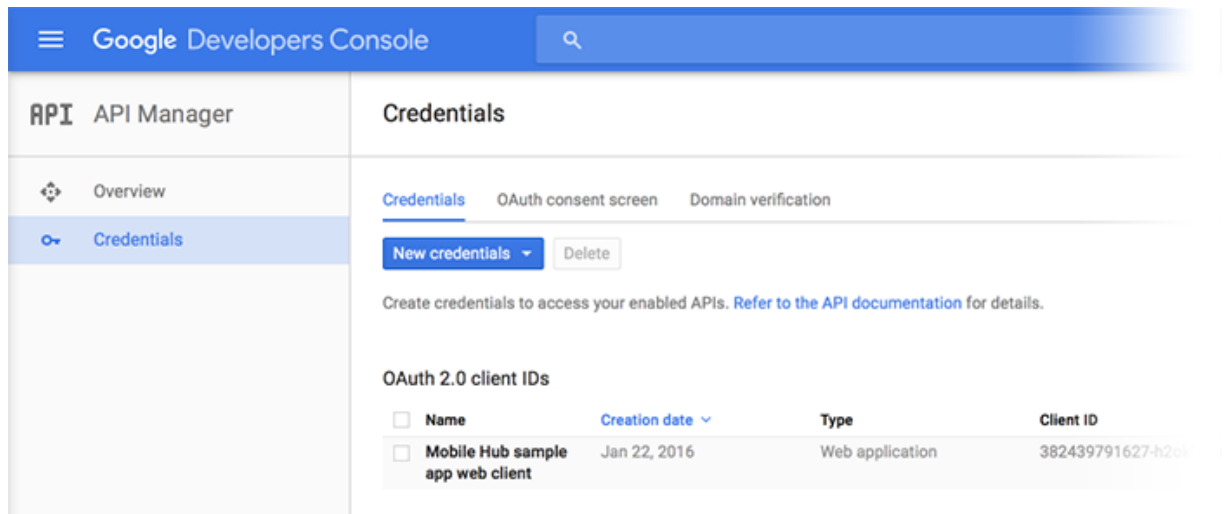
### To create an OAuth Android client ID

1. Go to the Google Developers Console at <https://console.developers.google.com>.

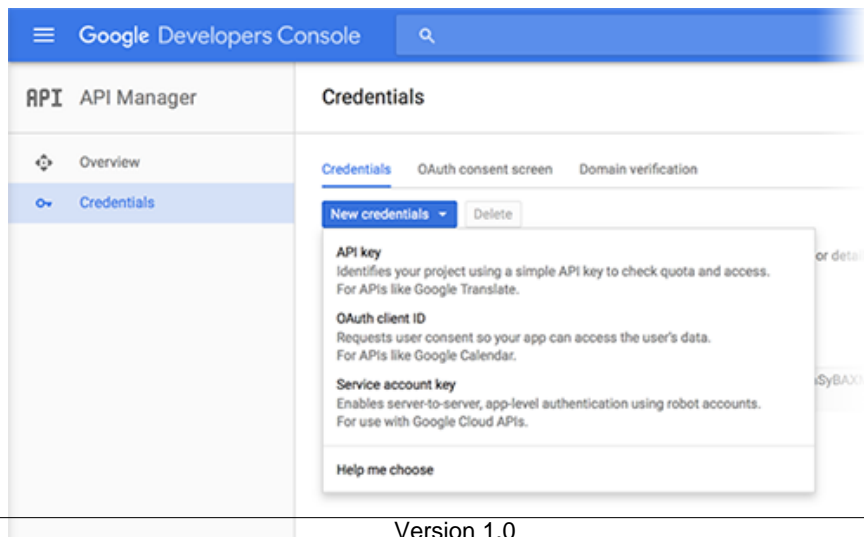
2. In the **Dashboard** for your project, go to the **Use Google APIs** section and then choose **Enable and manage APIs**.



3. In the API Manager, choose **Credentials** in the left side menu.

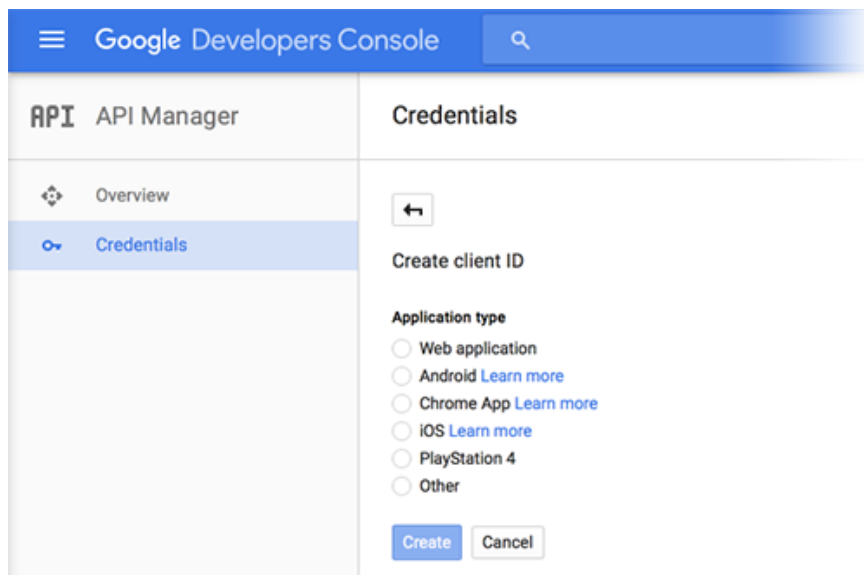


4. Choose **New credentials** and then choose **OAuth client ID**.





5. In **Create client ID**, choose **Android**.



6. In **Name**, enter a name in the format *com.amazon.mysampleapp Android client ID*.
7. In **Signing-certificate fingerprint**, enter the SHA-1 fingerprint. For more information about Google's process for obtaining your SHA-1 fingerprint, see [this Google support article](#).

## Credentials

←

Create client ID

**Application type**

Web application

Android [Learn more](#)

Chrome App [Learn more](#)

iOS [Learn more](#)

PlayStation 4

Other

**Name**

**Signing-certificate fingerprint**

Android devices send API requests directly to Google. Google verifies that each request comes from an Android app that matches a package name and SHA-1 signing-certificate fingerprint that you provide. Use the following command to get the fingerprint. [Learn more](#)

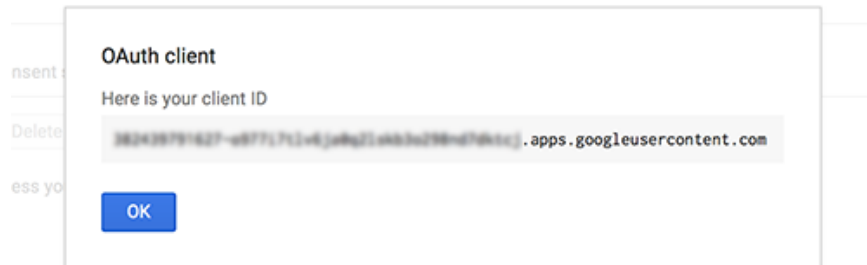
```
keytool -exportcert -alias androiddebugkey -keystore path-to-debug-or-production-keystore -list -v
```

**Package name**

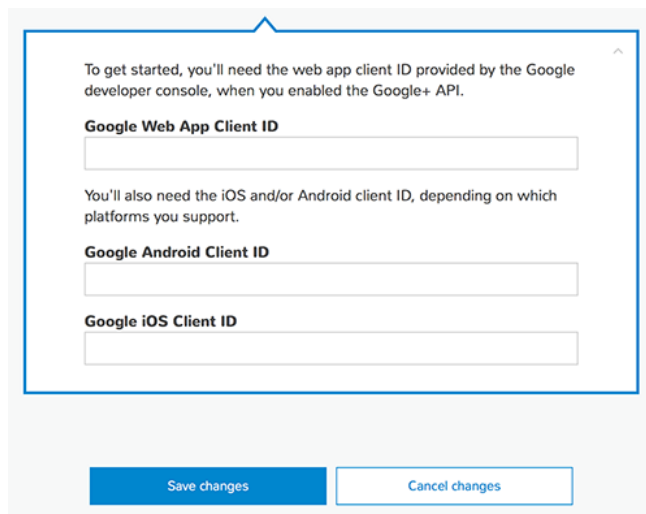
From your AndroidManifest.xml file

8. In **Package name**, enter the package name in the format *com.amazon.mysampleapp*.

9. Choose **Create**.
10. In the **OAuth client** pop-up, copy and save the value generated for your Android client ID. You will need this client ID to implement Google Sign-In in your Mobile Hub app. After you copy the client ID, choose **OK**.



11. Paste the Android client ID value into the Mobile Hub **Google Android Client ID** field for your project.



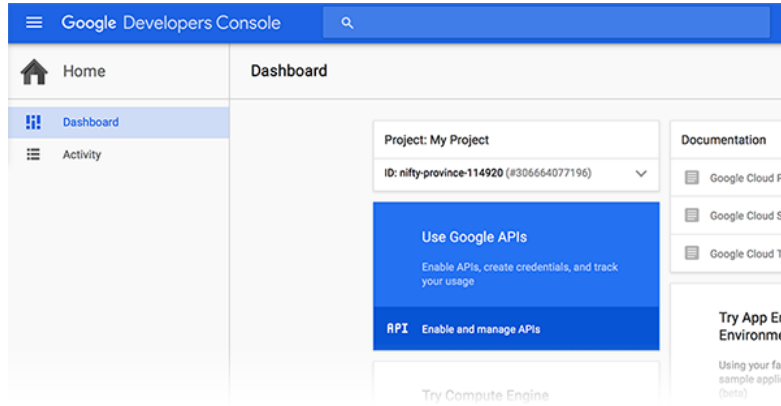
## Creating an OAuth iOS Client ID

To enable Google Sign-In for your iOS Mobile Hub sample app, you must create an iOS OAuth client ID for your app. This enables your Mobile Hub sample app to access Google APIs directly and to manage token lifecycle through Amazon Cognito Identity. This iOS OAuth client ID is in addition to the web application OAuth client ID that you created while [Creating a Google Developers Project and OAuth Web Client ID](#) (p. 60).

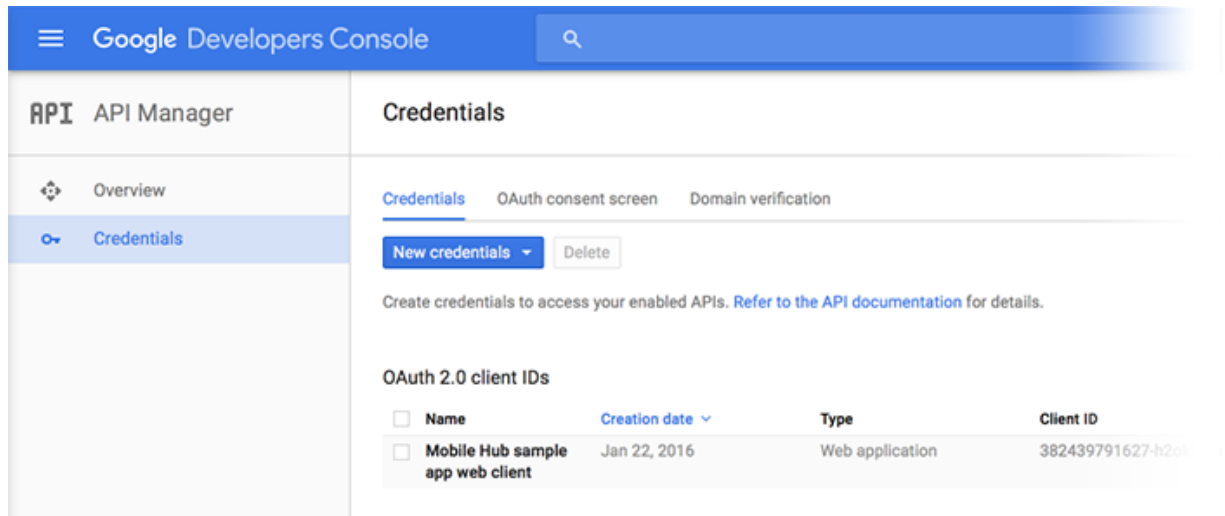
To integrate Google Sign-In for your iOS sample app, you must generate an iOS OAuth client ID in the Google Developers Console. You will provide this client ID to Mobile Hub during the Google Sign-In configuration.

### To create an OAuth iOS client ID

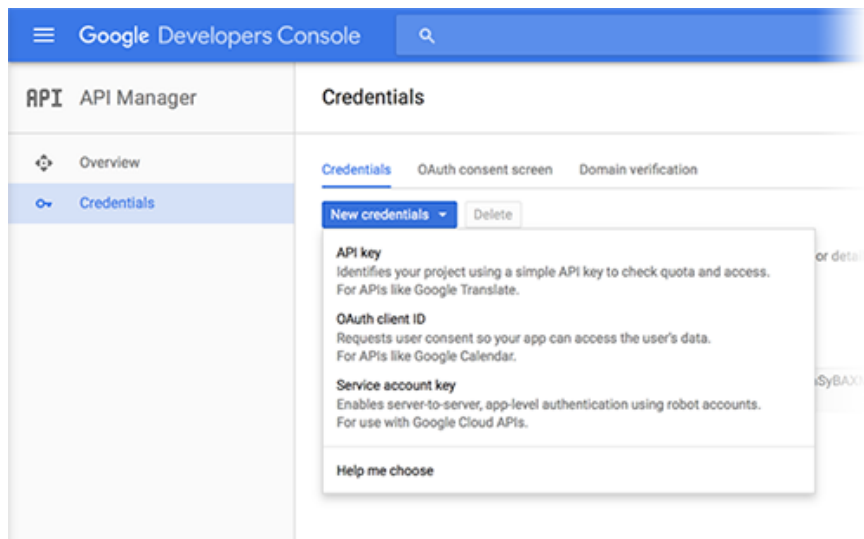
1. Go to the Google Developers Console at <https://console.developers.google.com>.
2. In the **Dashboard** for your project, go to the **Use Google APIs** section and then choose **Enable and manage APIs**.



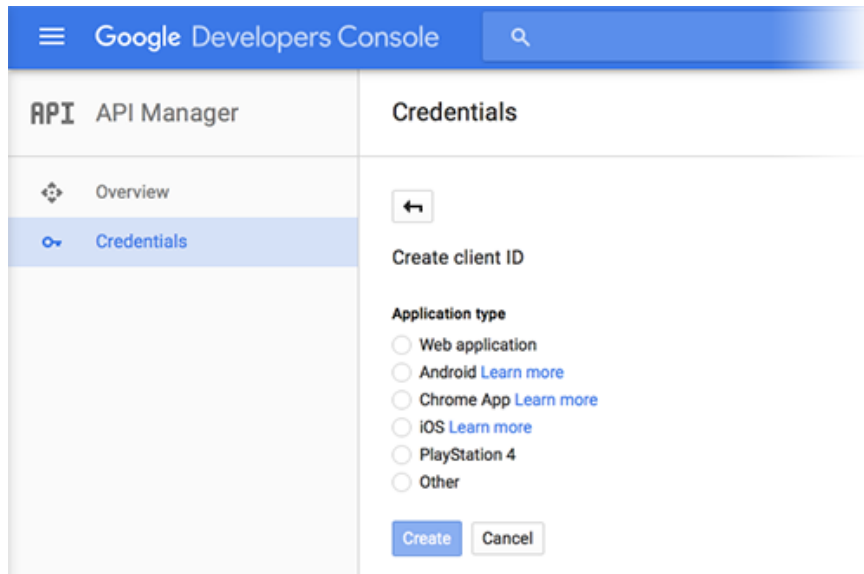
3. In the API Manager, choose **Credentials** in the left side menu.



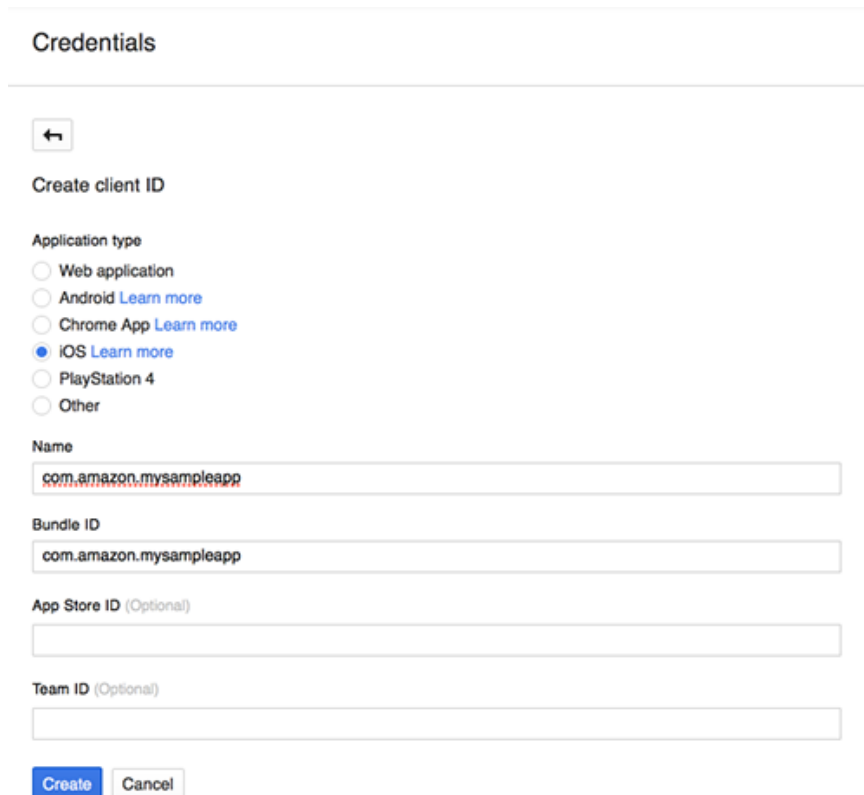
4. Choose **New Credentials** and then choose **OAuth client ID**.



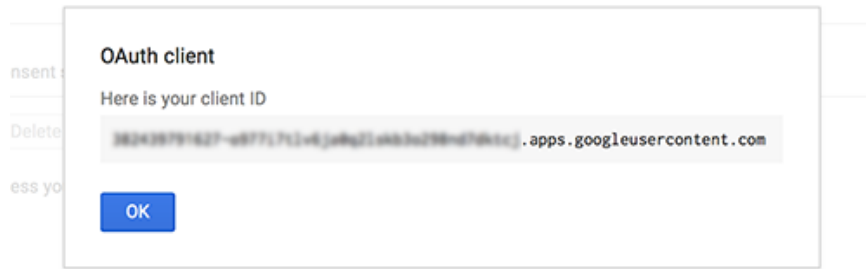
5. In **Create client ID**, choose **iOS**.



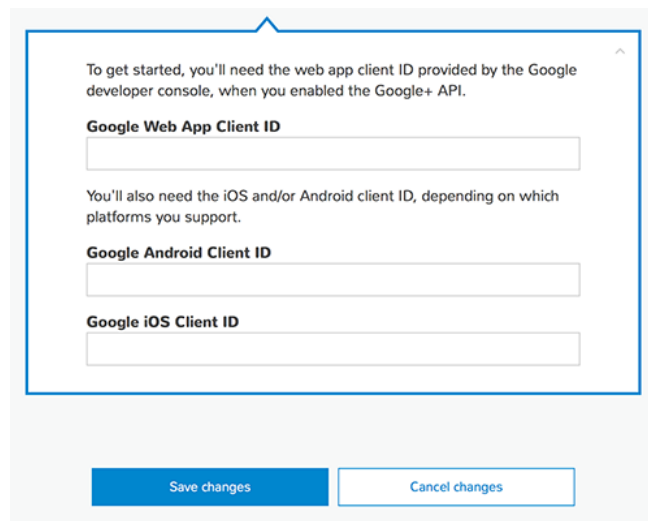
6. In **Name**, enter a name in the format *com.amazon.mysampleapp ios client ID*.
7. In **Bundle ID**, enter the bundle name in the format *com.amazon.mysampleapp*.



8. Choose **Create**.
9. In the **OAuth client** pop-up, copy and save the value that was generated for your iOS client ID. You will need these values to implement Google Sign-In in your Mobile Hub app. After you copy the client ID, choose **OK**.

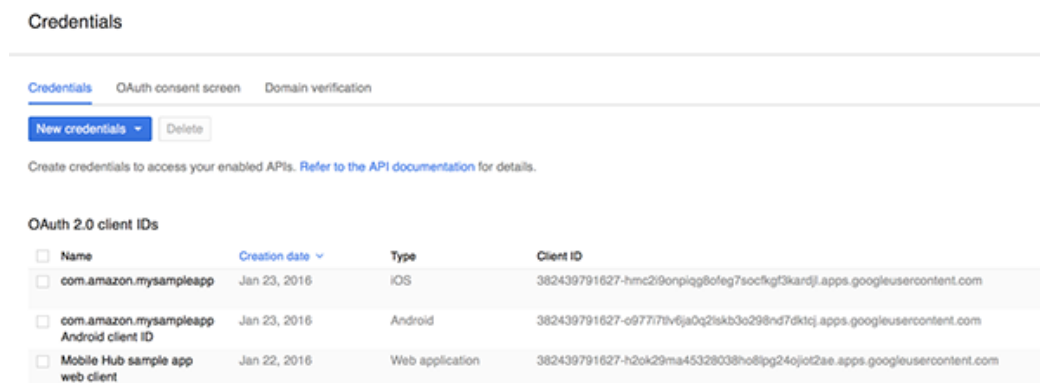


10. Paste the iOS client ID value into the Mobile Hub **Google iOS Client ID** field for your project.



## Verifying All Platform Client IDs

If your app supports both Android and iOS platforms, then your app project in the Google Developers Console will now have three client IDs: one for web application, one for Android, and one for iOS. You can verify that you have all of the credentials for all of the platforms by looking at the **Credentials** panel in the API Manager for your app, as shown in the following.



## Setting Up Custom Authentication

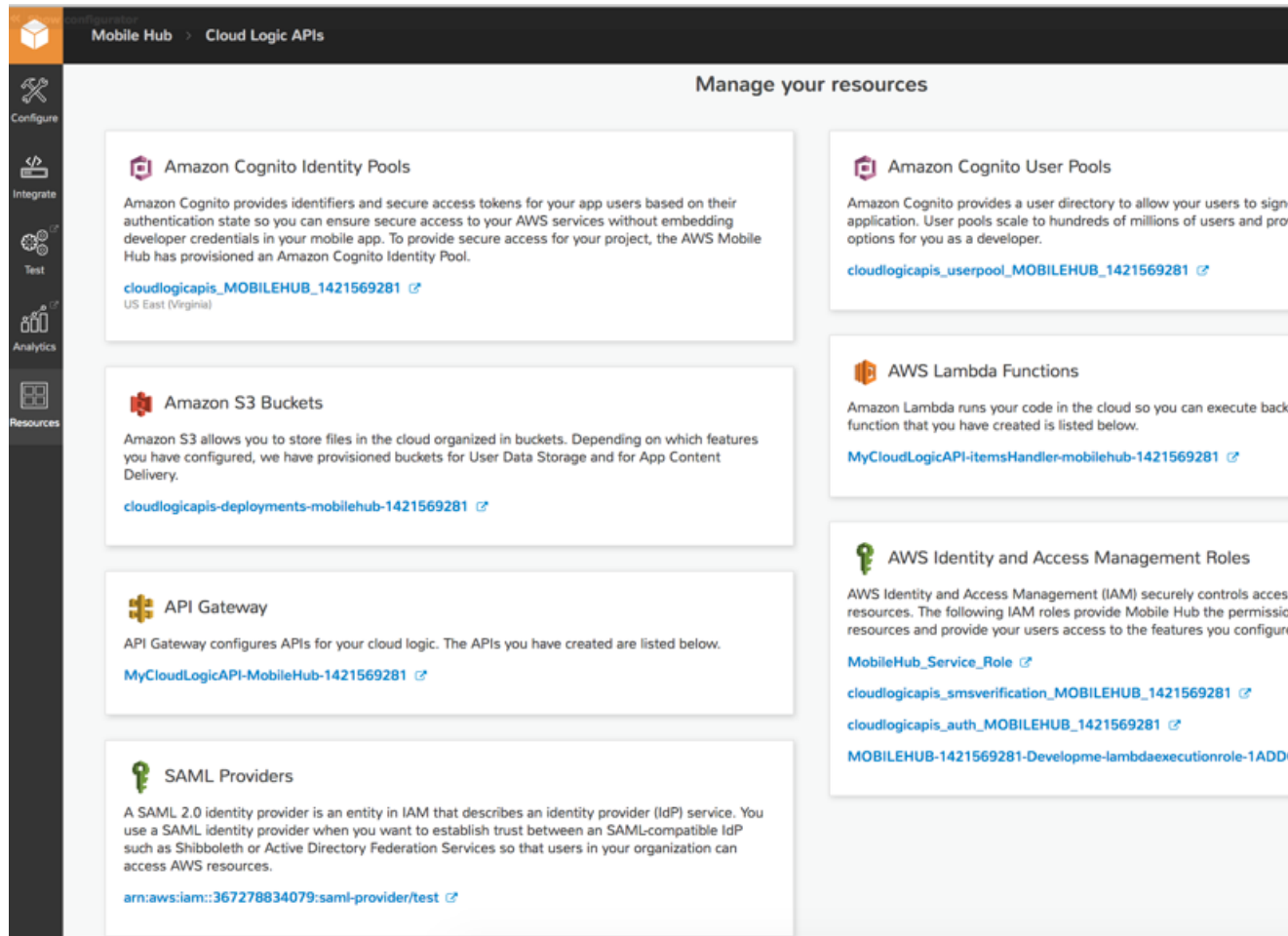
You can use your own authentication system, rather than identity federation provided by Facebook or Google, to register and authenticate your customers. The use of developer-authenticated identities involves interaction between the end-user device, your authentication back end, and Amazon Cognito. For more information, see the following blog entries:

- [Understanding Amazon Cognito Authentication](#)
- [Understanding Amazon Cognito Authentication Part 2: Developer-Authenticated Identities](#)

To use your own authentication system, you must implement an identity provider by extending the `AWSAbstractCognitoIdentityProvider` class and associating your provider with an Amazon Cognito identity pool. For more information, see [Developer Authenticated Identities](#) in the Amazon Cognito Developer Guide.

## Viewing AWS Resources Provisioned for this Feature

The following image shows the Mobile Hub **Resources** pane displaying elements typically provisioned for the User Sign-in feature.



## Quickstart App Details

In the Mobile Hub quickstart app, the User Sign-in demo enables users to use features that access AWS resources without authentication or by signing in to the app via identity providers including Facebook, Google, SAML Federation or Email and Password.

When you add User Sign-in to your project with the **Optional Sign-in** option, choosing the app's quickstart sign-in demo returns and displays the user's Amazon Cognito Identity Pool ID. This identifier is associated with the app instance's device currently accessing AWS resources.

When you add User Sign-in to your project with **Required Sign-in**, choosing the app's quickstart sign-in demo displays a sign-in experience branded to match the identity provider(s) configured in the project. Signing in to the demo authenticates the user in the selected identity provider service and returns and displays the Amazon Cognito Identity Pool ID identifier of the user.

# Build a Social Messaging App Using Multiple Mobile Hub Features

---

Amazon Web Services (AWS) provides mobile app developers with a powerful back-end platform that is easy to provision and easy to maintain. AWS provides your application with rich, flexible features that can scale up to meet very large demands and you pay only for the services you use.

In this walkthrough, we show you how to use AWS Mobile Hub to create the cloud backend for a social messaging application that we call AWSSampleMessenger. You will then complete a few integration steps and produce a demonstration chat app with features similar to those found in many popular mobile apps on either Android (Java) or iOS (Swift). This walkthrough takes about an hour to complete.

## Topics

- [Setup and Requirements \(p. 74\)](#)
- [Sample App Features \(p. 75\)](#)
- [Step 1: Creating a Mobile Hub Project \(p. 77\)](#)
- [Step 2: Adding User Sign-in \(p. 78\)](#)
- [Step 3: Adding NoSQL Database \(p. 79\)](#)
- [Step 4: Adding User Data Storage \(p. 83\)](#)
- [Step 5: Adding Push Notifications \(p. 84\)](#)
- [Step 6: Integrating Backend Features \(p. 86\)](#)
- [Sample App Code Tour \(p. 90\)](#)

## Setup and Requirements

To complete this walkthrough, you need the following.

### Prerequisites

You must have these things before beginning this walkthrough.

- Basic knowledge of app development using Android and Java or iOS and Swift



- AWS account

If you do not have an AWS account, use the following procedure to create one.

### To sign up for AWS

1. Open <http://aws.amazon.com/> and choose **Create an AWS Account**.
2. Follow the online instructions.

## Android System Requirements

To complete this walkthrough for Android apps, you need the following software.

- Android Studio 1.4 or newer
- Android SDK 4.4 (KitKat) API level 19 or newer
- Android SDK Build-tools 23.0.1
- Android device with Android OS 4.0.3 (Ice Cream Sandwich) API level 15 or newer for testing. Optionally, you can use [AWS Device Farm](#) to test your app remotely on real hardware.

## iOS System Requirements

To complete this walkthrough for iOS apps, you need the following software.

- Xcode 7.1 or newer version

## Sample App Features

The features of the AWSSampleMessenger app described in this walkthrough are typical of popular social messaging mobile apps. They enable the app to:

- Provide user login, using Facebook or Google+ accounts
- Enable creation of chat rooms and invitation to friends from the address book
- Send and receive messages
- Store and share pictures

## App Design and Implementation

Both front-end and back-end implementation of these features often requires substantial development work common to other kinds of mobile apps and back ends. This work includes:

- Creating sign-in functionality for each social media platform.
- Creating and maintaining an affordable and scalable cloud infrastructure that is able to handle millions of chat rooms and billions of messages every day.
- Capturing usage data, analysis and visualization to understand app performance and keep track of consumer use patterns. This data helps prioritize enhancements for ease-of-use and to maximize revenue.

As this walkthrough shows, AWS Mobile Hub can significantly reduce the development time required to build both the front end and back end of these features to publish a social media mobile app.

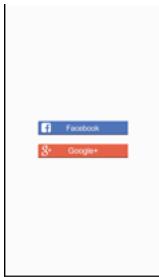
**Remember: this is a tutorial demo app.**

The design of the demo app reflects a balance between using best practices for concerns like security and user experience versus keeping the tutorial simple enough to complete in about an hour. To deploy this mobile app in production, we recommend you consider security requirements to meet your intended use cases, which are outside the scope of this simple project. For example, some enhancements could be made around:

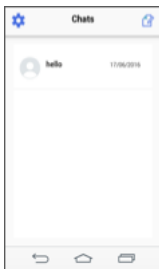
- **Access control** - Set up access permissions policies that provide users the ability to control who can contact them through the app.
- **Data encryption** - For the demo app, all chat communication between devices is encrypted in transport via https. However, the images shared via Amazon S3 and conversation text stored in Amazon DynamoDB are not encrypted by default in the sample app.
  - To implement server-side encryption of data in Amazon S3, see [Protecting Data Using Encryption](#) .
  - To learn about DynamoDB client-side encryption via a Java-based DynamoDB encryption client, see [Client-side Encryption for Amazon DynamoDB](#) .

## Using the App

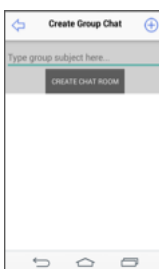
Upon starting the app, the user is given the option to sign in via Facebook or Google+.



Users create a new chat room by pressing the icon on the top-right corner of the screen.



Users invite others into the chat room from the address book by pressing the “+” icon on the top-right corner of the screen.



## Step 1: Creating a Mobile Hub Project

In this step, you create a Mobile Hub project called `AWSSampleMessenger`. As you perform the steps described in this walkthrough, Mobile Hub provisions and configures the AWS services and resources that make up the back-end features you add to your project.

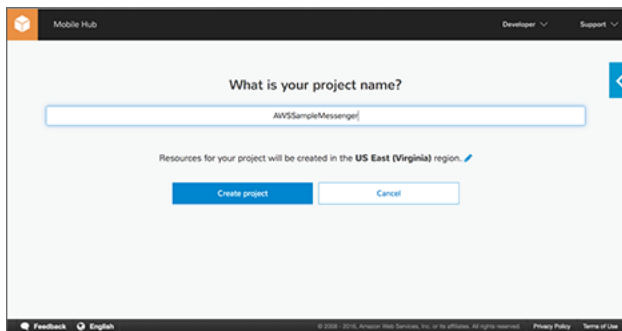
After you finish configuring all the features used by the project, you can download sample app code from Mobile Hub containing the identifiers of all resources created for those features. You then integrate the Mobile Hub project features into your `AWSSampleMessenger Getting Started` project, by copying those identifiers into its source code.

### To create the `AWSSampleMessenger` project in Mobile Hub

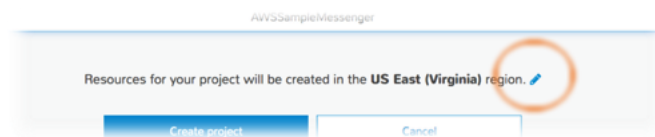
1. Navigate to the [AWS Mobile Hub console](#).
2. Choose **Create new mobile project**.



3. Type `AWSSampleMessenger` in the field labeled **What is your project name?**



4. Select a region by choosing the pencil icon to the right of the label that describes the default region where your project's AWS resources are hosted. Unless you need to select a different region, skip this step and choose **Create project**.



Reasons for selecting a region other than your default can include that your app is aimed at users in a region different than yours, or that your app depends on region-specific resources like AWS Lambda functions or Amazon SNS notification topics.

5. Choose **Create project** to create your Mobile Hub project and view the features you can add to it.

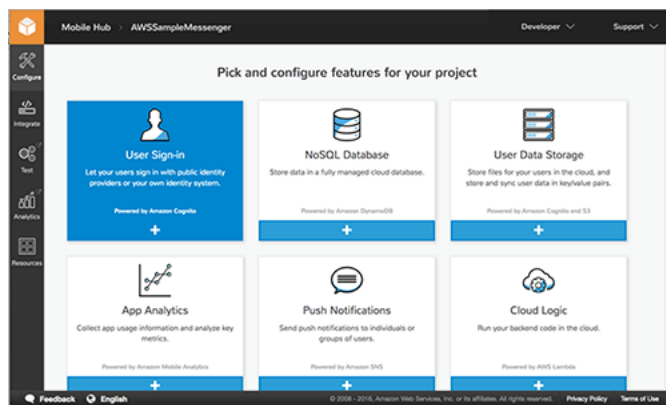
## Step 2: Adding User Sign-in

In this step you add the Mobile Hub User Sign-in feature to the AWSSampleMessenger project. The User Sign-in feature lets users sign in using their existing social network accounts, such as Facebook and Google+.

This feature uses Amazon Cognito Identity and AWS Identity and Access Management (IAM) services in conjunction with external identity providers. For more information about this Mobile Hub feature and the AWS services that it uses, see [User Sign-in \(p. 52\)](#).

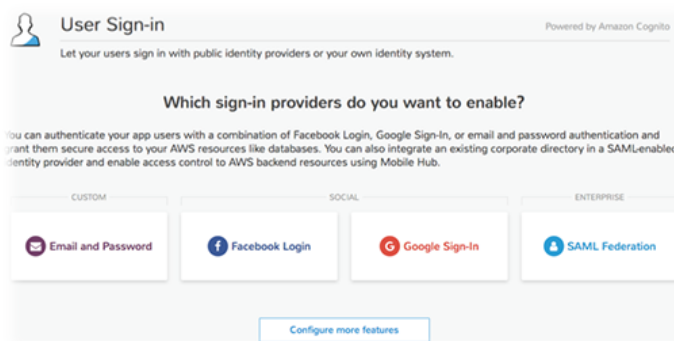
### To add the User Sign-in feature to the AWSSampleMessenger project

1. Choose **User Sign-in** in the Mobile Hub console.



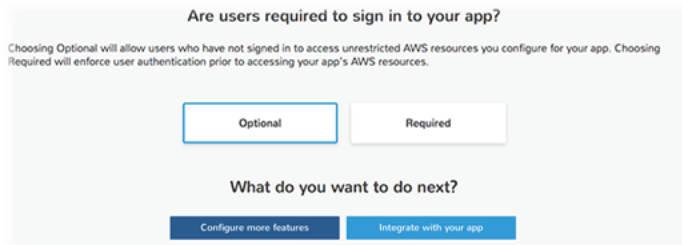
2. Configure the sign-in options for your app by choosing Facebook Login or Google Sign-in as a sign-in provider to authenticate user credentials from the provider. Then enter the identifiers from the service your app will use for sign-in. Choose **Save Changes**.

The AWSSampleMessenger project does not support SAML Federation or Email and Password sign-in features.



For more information about acquiring a Facebook app ID, see [Setting Up Facebook Authentication \(p. 57\)](#). For information about acquiring Google+ app and client IDs, see [Setting Up Google Authentication \(p. 60\)](#).

3. Choose **Required** and then choose **Configure more features**. Choosing required sign-in means that the quickstart app Mobile Hub generates requires user authentication to access all features of the app. With optional sign-in, some features are available to unauthenticated users.



## Step 3: Adding NoSQL Database

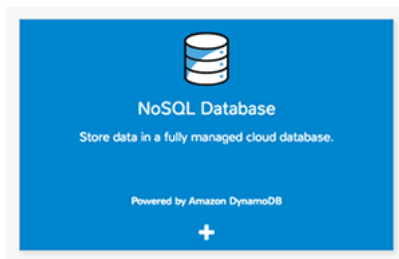
The Mobile Hub NoSQL Database feature uses Amazon DynamoDB to provide easily programmed, mobile-friendly data. NoSQL avoids many constraints of relational databases and can be scaled to meet AWSSampleMessenger app requirements for chat rooms created and used daily by millions of users. Data about chat rooms, conversations, and chat room recipients is stored in and retrieved from the cloud, using NoSQL Database tables.

The steps in this section implement the schema of the AWSSampleMessenger app in the form of DynamoDB tables that are provisioned and configured with access permissions for the users of your app. To learn more about how the schema is used in this project, see [AWSSampleMessenger schema and database code for iOS or Android](#).

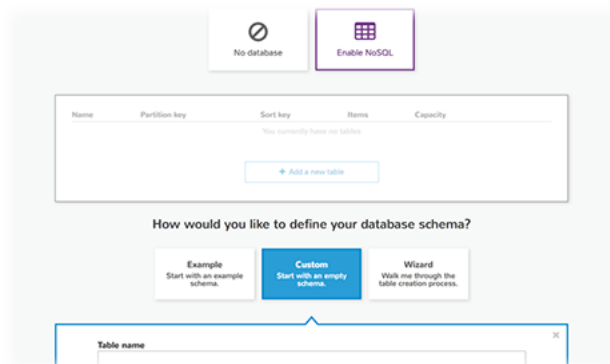
For more information about this Mobile Hub feature and the AWS services that it uses, see [NoSQL Database \(p. 26\)](#).

### To create NoSQL Database schema and tables for this app

1. Choose NoSQL Database from the Mobile Hub console.



2. Choose **Enable NoSQL**, then **+ Add a new table**, and then **Custom** to configure a new schema.



- To create the UserProfile table, type `UserProfile` for the **Table name** and then choose **Protected** permissions. You use this table to contain data for chat participant identities and relationships.

Table name: UserProfile

Table resource: awssamplemessengermobilehub-1997985270-UserProfile

What permissions would you like for this table?

Public: Any app user can read and write to any item.

Protected: Any app user can read, only owner can write to item. (Selected)

Private: Only the owner can read and write the item.

What attributes do you want on this table?

Each table has a built-in Primary Index which must have a Partition key and optionally may have a Sort key. Key attributes must be of type string, number or binary. Non-key attributes may be added here or from within your mobile app code.

Attribute name	Type	Partition key	Sort key
userId	string	<input checked="" type="checkbox"/>	<input type="checkbox"/>

- Choose **Add attribute** to add the following attributes to the table.
  - Attribute name: `phone`; Type: string; Sort key: Yes (selected)
  - Attribute name: `name`; Type: string; Sort key: No
  - Attribute name: `pushTargetArn`; Type: string set; Sort key: No

What attributes do you want on this table?

Each table has a built-in Primary Index which must have a Partition key and optionally may have a Sort key. Key attributes must be of type string, number or binary. Non-key attributes may be added here or from within your mobile app code.

Attribute name	Type	Partition key	Sort key
userId	string	<input checked="" type="checkbox"/>	<input type="checkbox"/>
phone	string	<input type="checkbox"/>	<input checked="" type="checkbox"/>
name	string	<input type="checkbox"/>	<input type="checkbox"/>
pushTargetArn	string set	<input type="checkbox"/>	<input type="checkbox"/>

Queries this table can perform.

Primary Index Queries	Examples
Get Item	Find item with <code>userId = ABC</code> and <code>phone = ABC</code>
Query by Partition Key	Find all items with <code>userId = ABC</code>
Query by Partition Key and Sort Condition	Find all items with <code>userId = ABC</code> and <code>phone is &lt; zzz</code>
Query by Partition Key, Sort Condition, and Filter	Find all items with <code>userId = ABC</code> and <code>phone is between aaa and zzz</code> and <code>Name begins with XY</code>

- Choose **Create table**.

pushTargetArn string set

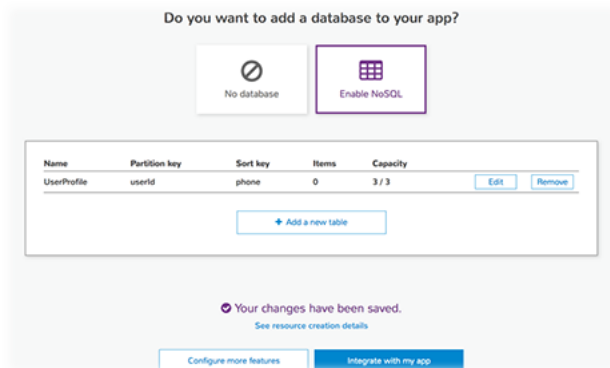
Queries this table can perform.

Primary Index Queries	Examples
Get Item	Find item with <code>userId = ABC</code> and <code>phone = ABC</code>
Query by Partition Key	Find all items with <code>userId = ABC</code>
Query by Partition Key and Sort Condition	Find all items with <code>userId = ABC</code> and <code>phone is &lt; zzz</code>
Query by Partition Key, Sort Condition, and Filter	Find all items with <code>userId = ABC</code> and <code>phone is between aaa and zzz</code> and <code>Name begins with XY</code>

What indexes do you want on this table?

Create table

- Choose **Create table** in the confirmation dialog.
- Now create the ChatRoom table. Choose **Add a new table** then choose **Custom**. Type `ChatRoom` for the **Table name** and then choose **Protected** permissions.

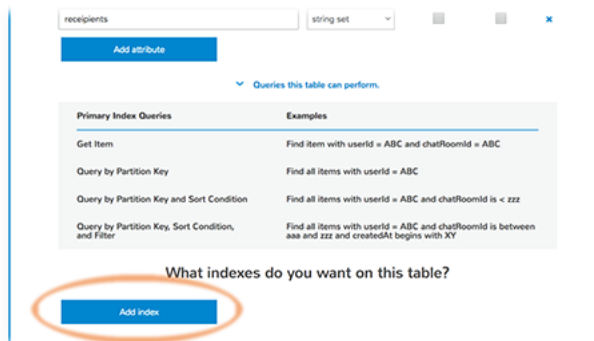


8. Choose **Add attribute** to add each of the following attributes to the table.

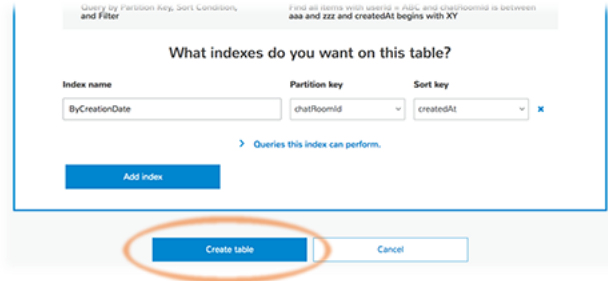
- Attribute name: chatRoomId; Type: string; Sort key: Yes (selected)
- Attribute name: createdAt; Type: string; Sort key: No
- Attribute name: name; Type: string; Sort key: No
- Attribute name: recipients; Type: string set; Sort key: No



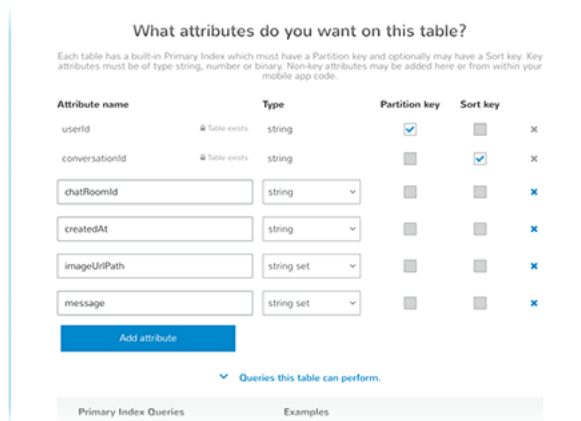
9. Choose **Add index**.



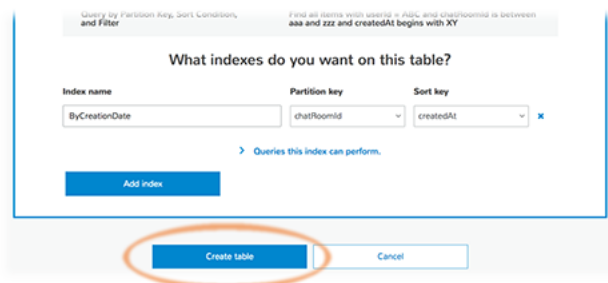
10. Type `ByCreationDate` for **Index name**. Choose `chatRoomId` for **Partition key** and `createdAt` for **Sort key**.



11. Choose **Create table** and then choose **Create table** in the confirmation dialog.
12. Now create the Conversation table. Choose **Add a new table** and then choose **Custom**. Type **conversation** for **Table name** and then choose **Protected** permissions.
13. Choose **Add attribute** to add each of the following attributes to the table.
  - Attribute name: `conversationId`; Type: string; Sort key: Yes (selected)
  - Attribute name: `createdAt`; Type: string; Sort key: No
  - Attribute name: `chatRoomId`; Type: string; Sort key: No
  - Attribute name: `imageUrlPath`; Type: string set; Sort key: No
  - Attribute name: `message`; Type: string set; Sort key: No

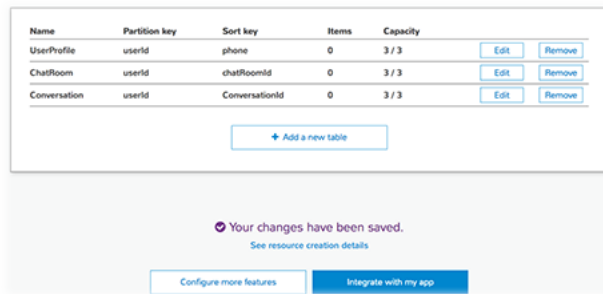


14. Choose **Add index**.
15. Type `ByCreationDate` for **Index name**, choose `chatRoomId` for **Partition key** and `createdAt` for **Sort key**.



16. Choose **Create table** and then choose **Create table** in the confirmation dialog.





17. Choose **Configure more features**.

## Step 4: Adding User Data Storage

The Mobile Hub User Data Storage feature enables your mobile app user to store files and synchronize app data in the cloud. The AWSSampleMessenger app uses User Data Storage to create an Amazon S3 bucket that acts as the storage location for images exchanged during a conversation.

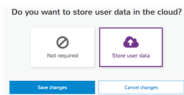
For more information about User Data Storage and the AWS services that it uses, see [User Data Storage](#) (p. 46).

### To add User Data Storage to the project

1. Choose User Data Storage on the Mobile Hub console.



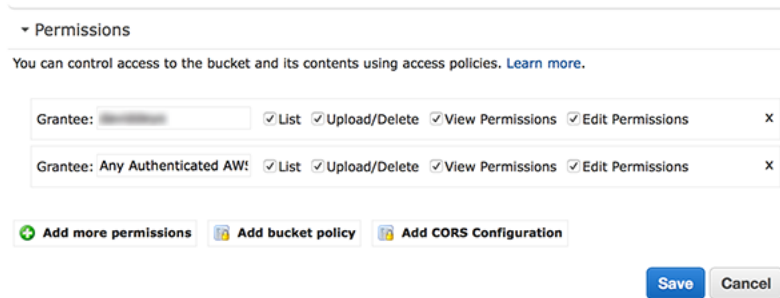
2. Choose **Store user data** and then choose **Save changes**.



3. Open the S3 console at <https://console.aws.amazon.com/s3/> to modify the access policy for the bucket you created. The access policy you are modifying ensures that only signed-in users have access and that only the user who posts the image can delete it.
4. Choose the bucket Mobile Hub created for your project. The name looks like:

```
awssamplemesenger-userfiles-mobilehub-0123456789
```

5. Choose **Properties** from the top-right corner.
6. In **Permissions**, choose **+ Add more permissions**.
7. Choose **Any Authenticated AWS User** from the **Grantee** drop-down list.



▼ Permissions

You can control access to the bucket and its contents using access policies. [Learn more.](#)

Grantee: [REDACTED]	<input checked="" type="checkbox"/> List	<input checked="" type="checkbox"/> Upload/Delete	<input checked="" type="checkbox"/> View Permissions	<input checked="" type="checkbox"/> Edit Permissions	X
Grantee: Any Authenticated AWS User	<input checked="" type="checkbox"/> List	<input checked="" type="checkbox"/> Upload/Delete	<input checked="" type="checkbox"/> View Permissions	<input checked="" type="checkbox"/> Edit Permissions	X

[Add more permissions](#) [Add bucket policy](#) [Add CORS Configuration](#)

**Save** **Cancel**

8. Choose **Save**.
9. Return to the Mobile Hub console and choose **Configure more features**.

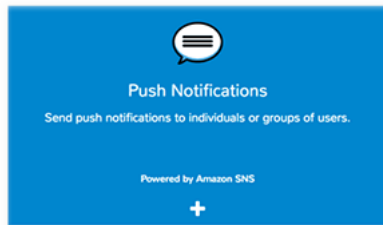
## Step 5: Adding Push Notifications

The Mobile Hub Push Notifications feature uses Amazon Simple Notification Service (Amazon SNS) to provide robust messaging services for mobile apps. AWSSampleMessenger uses this feature as the mechanism for notifying app users of incoming messages.

For more information about the Mobile Hub Push Notifications feature, see [Push Notifications \(p. 32\)](#).

### To add Push Notifications to your Mobile Hub project

1. Choose Push Notifications on the Mobile Hub console.



2. Choose **Enable Push Notifications**
3. Choose the platform for the project.

Choose **Android** to configure push notifications for Android applications. For information on obtaining your API key and sender ID, see [Setting Up Android Push Notification \(p. 41\)](#).

Do you want to send push notifications to your app?

Not required  Enable push

What platforms do you want to send messages to?

Android  iOS

To enable Google Cloud Messaging (GCM), you will need to enter your Google credentials here.

**API Key**

**Sender ID**

**For iOS:** Choose **iOS (Apple Push Notification service)** to configure push notifications for iOS applications. For information on obtaining your Apple Push Certificate (.p12), see [Setting Up iOS Push Notification \(p. 34\)](#).

Do you want to send push notifications to your app?

Not required  Enable push

What platforms do you want to send messages to?

Android  iOS

To enable Apple Push Notification service (APNs), you'll need to provide your Universal Apple Certificate here.

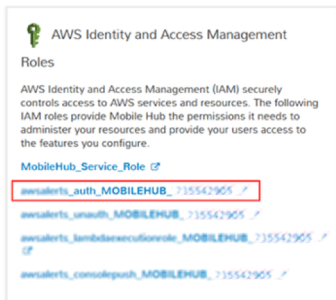
**P12 Certificate**

No file selected.

**Certificate Password**

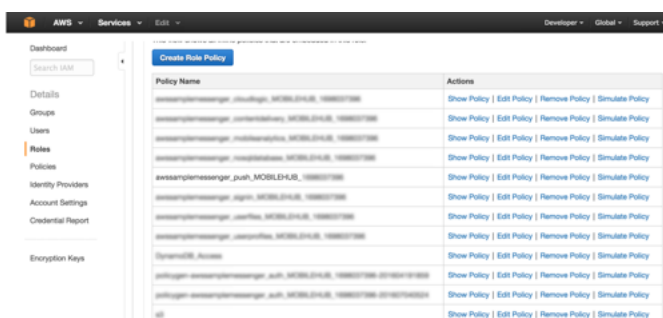
Optional

4. Choose **Save changes**.
5. Add push notification publishing permissions. To send push notifications to your Amazon SNS platform applications, modify the authenticated IAM role for your project so its permissions policy allows publishing messages from the app.
6. Choose your project in the Mobile Hub console and then choose **Resources** from the left panel.
7. Choose your authenticated (`_auth_`) role from the AWS Identity and Access Management card to launch the IAM console where you can edit the policy associated with this role.



- In the list of policies attached to the role, choose **Edit Policy** for the push policy with a name similar to the following.

awssamplemessenger\_push\_MOBILEHUB\_012345678



- Check that the `sns:Publish` action is present in the first statement for publishing messages to any of your SNS applications. This allows authenticated app users to publish to other SNS platform endpoints (other registered devices).



- Choose **Apply Policy**.
- Return to the Mobile Hub console.

## Step 6: Integrating Backend Features

This walkthrough guides you through the Mobile Hub process of configuring features for your AWSSampleMessenger project. This process creates and configures new services and their resources in the AWS cloud for your project. Mobile Hub offers two options for integrating those services into your mobile app:

- Integrate these features into your mobile app by following instructions for incorporating and invoking each feature you configure in an existing mobile app you have created
- Use a Mobile Hub sample app with these features by downloading an auto-generated project containing iOS or Android code that you can build and demonstrate use of each of the features you configured.

The AWSSampleMessenger project is based on a project auto-generated by Mobile Hub.

#### Topics

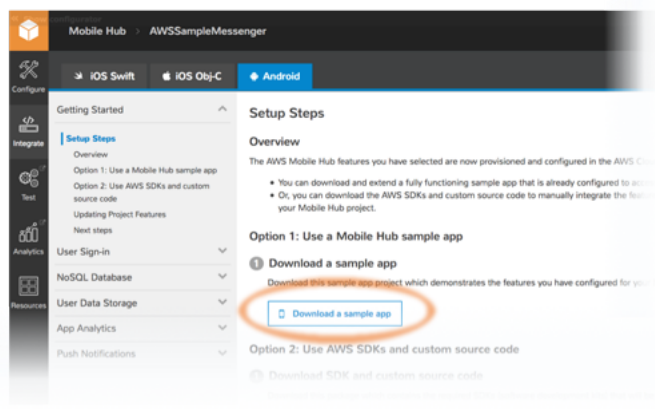
- [Integrating Android Apps \(p. 87\)](#)
- [Integrating iOS Apps \(p. 89\)](#)

## Integrating Android Apps

You will enable the AWS services that you configured for the AWSSampleMessenger app using the identifiers and attributes of the resources that Mobile Hub provisioned as you completed the previous steps. The AWSSampleMessenger sample project is structured so that you can quickly copy and paste that information from the downloadable Mobile Hub quickstart app.

### To integrate your AWS back-end services into an Android app

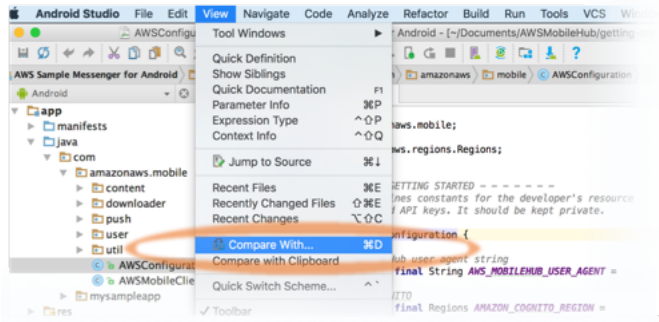
1. From your app, choose **Integrate** from the left navigation menu.
2. Choose **Android**.
3. In the **Getting Started** section, choose **Option 1: Use a Mobile Hub sample app**.
4. Choose **Download a sample app**.



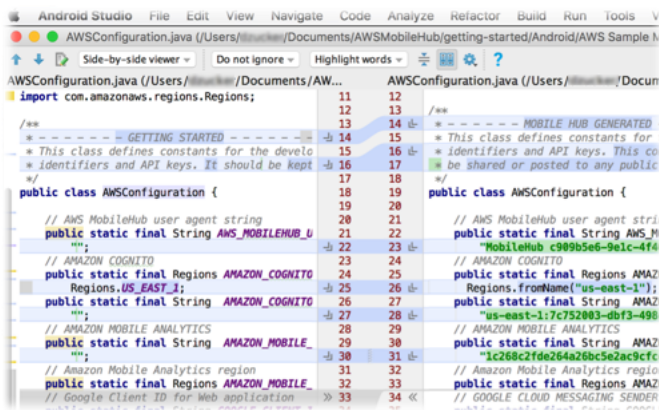
5. Download the Getting Started AWSSampleMessenger Android package and unzip it.

<https://s3-us-west-2.amazonaws.com/aws-mobile-hub/samples/awssamplemessenger/AWSMessengerAndroid.zip>

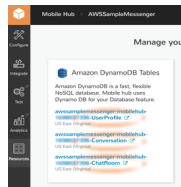
6. Open the app project in Android Studio and then open the `java/com/amazonaws/mobile/AWSConfiguration.java` file. Fill in the values for each item whose value is an empty string. You can find the correct values, which map to the AWS services you configured, in the `AWSConfiguration.java` file in the Mobile Hub project.
7. Choose the **View** menu and then choose **Compare With....**



- Find and select `AWSConfiguration.java` in the Mobile Hub-generated project. Copy values from the Mobile Hub project to the Getting Started project with the two files side by side.



- Verify the table names in the Amazon DynamoDB Tables section of the `AWSSampleMessenger` project resources in the Mobile Hub console. You need these table names in the next step.



- Add the following variable declarations to `AWSConfiguration.java` in your Getting started app.

```
//ChatRoomDO TABLE NAME
public static final String AMAZON_DYNAMODB_TABLENAME_CHATROOM = "<Your
ChatRoom Table Name>";
//ConversationDO TABLE NAME
public static final String AMAZON_DYNAMODB_TABLENAME_CONVERSATION = "<Your
Conversation Table Name>";
//UserProfileDO TABLE NAME
public static final String AMAZON_DYNAMODB_TABLENAME_USERPROFILE = "<Your
UserProfile Table Name>";
```

- Open the `res/values/strings.xml` file in the Getting Started project and insert your Facebook App ID as the value of `facebook_app_id`. Find your Facebook App ID in the Mobile Hub console by returning to the configuration page for the User Sign-in feature of this app. Choose **User Sign-in** and the Facebook icon.

```
<string name="facebook_app_id">0123456789012345</string>
```

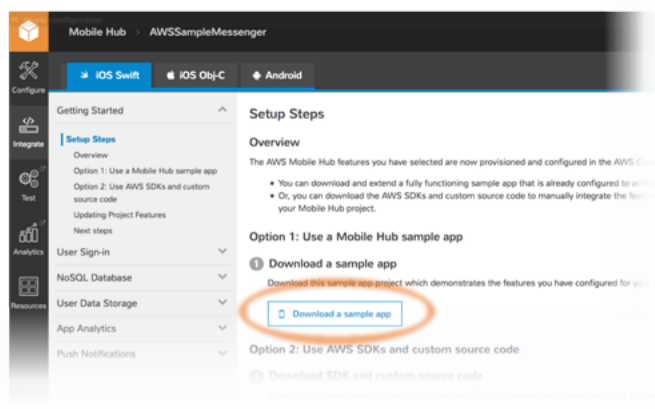
You have now completed integrating your AWS services backend into your Getting Started app.

## Integrating iOS Apps

You will enable the AWS back-end services that you configured for the AWSSampleMessenger app using the identifiers and attributes of the resources that Mobile Hub provisioned as you completed the previous steps. The AWSSampleMessenger sample project is structured so that you can quickly copy and paste that information from the downloadable Mobile Hub quickstart app.

### To integrate your AWS back-end services into an iOS app

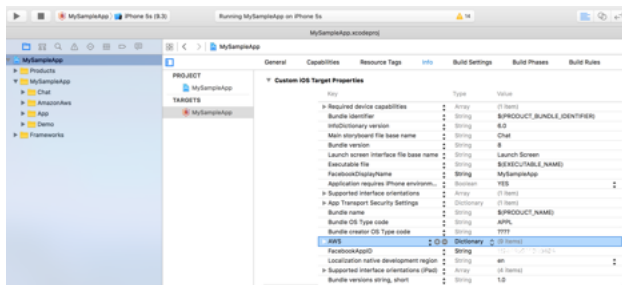
1. From your app, choose **Integrate** from the left navigation menu.
2. Choose **iOS**.
3. In the **Getting Started** section, choose **Option 1: Use a Mobile Hub sample app**.
4. Choose **Download a sample app**.



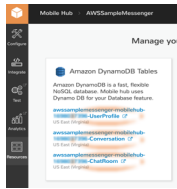
5. Download the Getting Started AWSSampleMessenger iOS package and open the project in Xcode.

<https://s3-us-west-2.amazonaws.com/aws-mobile-hub/samples/awssamplemessenger/AWSMessengerSwift.zip>

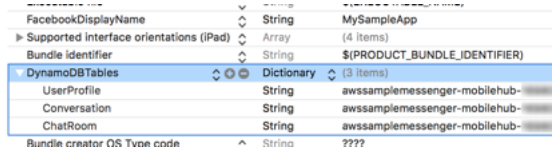
6. Open the Mobile Hub quickstart sample project in Xcode. Open the **Info** tab and copy the entire AWS dictionary.
7. Delete the AWS dictionary file of the `info.plist` in your Getting Started project and then paste the dictionary copied in the previous step.



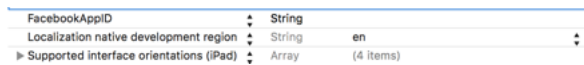
- Look up and note the table names given under the Amazon DynamoDB Tables section of the AWSSampleMessenger project resources in the Mobile Hub console. You need these table names in the next step.



- Enter the DynamoDB table names in the **DynamoDBTables** key section of the **Info** tab in the AWSSampleMessenger project.



- Enter your Facebook App ID in the Getting Started project's `FacebookAppID` key, found in the **Info** tab. Find your Facebook App ID in the Mobile Hub console by returning to the configuration page for the User Sign-in feature of this app. Choose **User Sign-in** and the Facebook icon.



- Copy the URL Schemes from the `info.plist` of your Mobile Hub project and paste them into the `info.plist` of your Getting Started project.



You have now integrated your AWS services backend into your Getting Started app.

## Sample App Code Tour

This section describes how AWS backend services configured for this mobile app are integrated and invoked in an app running either Android or iOS.

### Topics

- [Android Source Code \(p. 90\)](#)
- [iOS Source Code \(p. 103\)](#)

## Android Source Code

This section describes how the AWS back-end services configured for this Android mobile app are integrated and invoked.

### Topics



- [User Sign-in: Amazon Cognito and Identity Providers \(p. 91\)](#)
- [NoSQL Database: Amazon DynamoDB and the AWSSampleMessenger Schema \(p. 91\)](#)
- [User Data Storage: Amazon S3 \(p. 96\)](#)
- [Push Notifications: Amazon SNS \(p. 98\)](#)

## User Sign-in: Amazon Cognito and Identity Providers

AWSSampleMessenger uses Amazon Cognito Identity to implement the sign-in feature of your app.

### Sign-in Required

AWSSampleMessenger user sign-in is configured so that a user must provide credentials that are authenticated by an identity provider (Facebook Login or Google+ Sign-in, in this instance). When a user signs in, a three-way communication occurs between the mobile client, the identity provider, and Amazon Cognito. Successful authentication results in the user having an authenticated Amazon Cognito Identity in the Cognito identity pool, associated with your version of this app.

In configuring the User Sign-in feature, Mobile Hub provisions AWS Identity and Access Management (IAM) roles and policies regarding authenticated and unauthenticated access to the resources you configured, such as Amazon DynamoDB tables. For AWSSampleMessenger, a policy allowing access is attached to the authenticated role. That authenticated role is attached to the Amazon Cognito identity pool for your app, allowing any signed-in user access to the back-end resources for the app.

### User Sign-in Components

#### Mobile Client

The Mobile Client bootstraps the app. It creates an identity manager to establish the user identity with Amazon Cognito. It also enables the features you selected for the project. For example, it handles the push notification device tokens, and indicates to the Amazon Mobile Analytics client when to pause and resume metrics collection for the user session. Code for this component can be found in `java/com/amazonaws/mobile/AWSMobileClient.java`.

#### Identity Manager

The Identity Manager keeps track of the identity of the current app user by storing the user's Amazon Cognito identity pool ID. It assists with signing the user into the app if you have User Sign-in enabled in your project. It passes results of the sign-in operation back to the application's handler. Code for this component can be found in `java/com/amazonaws/mobile/user/IdentityManager.java`.

#### Sign-in Manager

The Sign-in Manager is a single component that creates the sign-in identity providers and orchestrates sign-in call flows. It is responsible for keeping track of the most recent provider, refreshing credentials, and initializing sign-in buttons with the sign-in providers. Code for this component can be found in `java/com/amazonaws/mobile/user/signin/SignInManager.java`.

## NoSQL Database: Amazon DynamoDB and the AWSSampleMessenger Schema

AWSSampleMessenger uses Amazon DynamoDB to implement the NoSQL database feature of your app.

The Amazon DynamoDB tables configured for this app give the cloud data structure where app data is dynamically stored in and retrieved from the `UserProfile`, `ChatRoom`, and `Conversation` tables.

## Table Indexes

Each table contains a primary index that consists of a primary key and a sort key, both of which organize data for most efficient query path. The `UserProfile` table is designed so that queries of `userId` and `phone` have the best performance. The `ChatRoom` and `Conversation` tables also have a secondary index called `ByCreationDate` to maximize query performance based on the chat room and the date when the chat room was created or the conversation happened.

## Data Relationships

The tables contain some attributes in common. These common attributes indicate to the app the relationship between items (equivalent to rows in a relational database). For instance, the recipients participating in a Chatroom can be enumerated for a given conversation by querying the `ChatRoom` table `recipients` attribute value based on the shared `chatRoomId` value.

## Security

The tables of this app are all *Protected*, which in Mobile Hub means the owner of an item in a table can read and modify an item, everyone else can read but not modify the item data. The owner, in this case, is the Amazon Cognito Identity ID connected to the instance of the app that created the item. This security is enforced in the following ways:

- When a user takes an action in the app that creates an item, their Amazon Cognito Identity ID is captured as the `userId` attribute value for the item.
- Amazon DynamoDB table permissions have been configured such that when a user takes an action that would modify the data of that item, the `userId` value is compared to the Amazon Cognito Identity ID of the active user. If these values match, then modifying the data is allowed.

## Schema

The following summarizes the table definitions of the `AWSSampleMessenger` schema.

Table Name: `UserProfile`

Table property	Type of value	Value
Security		Protected
Primary Key	string	<code>userId</code>
Sort Key	string	<code>phone</code>
Attribute	string	<code>name</code>
Attribute	string set	<code>pushTargetArn</code>

Table Name: `Chatroom`

Table property	Type of value	Value
Security		Protected
Primary Key	string	<code>userId</code>
Sort Key	string	<code>chatRoomId</code>
Attribute	string	<code>createdAt</code>

Table property	Type of value	Value
Attribute	string	name
Attribute	string set	recipients
Secondary Index Name		ByCreationDate
Secondary Index Primary Key	string	chatRoomId
Secondary Index Sort Key	string	createdAt

Table Name: Conversation

Table property	Type of value	Value
Security		Protected
Primary Key	string	userId
Sort Key	string	conversationId
Attribute	string	createdAt
Attribute	string	chatRoomId
Attribute	string	imageUrlPath
Attribute	string	message
Secondary Index Name		ByCreationDate
Secondary Index Primary Key	string	chatRoomId
Secondary Index Sort Key	string	createdAt

## How DynamoDB is Used

The following code in `CreateChatRoomActivity` uses the `AWSSampleMessenger` schema to let users create chat rooms and initiate conversations with one or multiple contacts from the device's address book.

An enumeration of selected recipients is retrieved from `loadUsersWithPhoneList` in the `UserProfile` table. Running this code loads all the contacts from the authenticated user's database.

```
public PaginatedScanList<ChatRoomDO>
loadUserChatRooms(CognitoCachingCredentialsProvider provider){

    ddbClient = new AmazonDynamoDBClient(provider.getCredentials());

    ddbClient.setRegion(Region.getRegion(AWSConfiguration.AMAZON_DYNAMODB_REGION));
    mapper = new DynamoDBMapper(ddbClient);

    //find current user chat room on the base of userId
    Map<String, AttributeValue> filter = new HashMap<String,
AttributeValue>();
    filter.put(":userId",new
AttributeValue().withS(provider.getCachedIdentityId()));
```

```

        DynamoDBScanExpression scanExpression = new DynamoDBScanExpression()
            .withFilterExpression("userId=:userId OR
contains(recipients, :userId)")
            .withExpressionAttributeValues(filter);

        PaginatedScanList<ChatRoomDO> scanResult = mapper.scan(ChatRoomDO.class,
scanExpression);

        if(scanResult != null){
            return scanResult;
        }

        return null;
    }

```

`saveNewChatRoom` inserts a new chat room in the `ChatRoom` table, which lets the user create a new chat room, with one or more recipients gleaned from the data produced by `loadUsersWithPhoneList`.

```

String groupName, Set<String> recipientId, String chatRoomId) {

    AmazonDynamoDBClient ddbClient = new
AmazonDynamoDBClient(provider.getCredentials());

    ddbClient.setRegion(Region.getRegion(AWSConfiguration.AMAZON_DYNAMODB_REGION));
    mapper = new DynamoDBMapper(ddbClient);

    chatRoomDO = new ChatRoomDO();
    chatRoomDO.setUserId(provider.getCachedIdentityId());
    chatRoomDO.setChatRoomId(chatRoomId);
    chatRoomDO.setCreatedAt(dt);
    chatRoomDO.setName(groupName);
    chatRoomDO.setRecipients(recipientId);
    mapper.save(chatRoomDO);
}

```

`createChatRoom` calls the two preceding functions in sequence.

```

public void createChatRoom(View view){
    if( contactsAdapter.phNoList.size(>0){

        new AsyncTask<Void, Void, String>() {
            @Override
            protected String doInBackground(Void... params) {
                String msg = "";
                try {
                    ChatUserProfileManager userProfileManager = new
ChatUserProfileManager();
                    awsRecipientUsers =
userProfileManager.loadUsersWithPhoneList(credentialsProvider,
contactsAdapter.phNoList.toArray());

                } catch (AmazonServiceException ex) {
                    msg = ex.getLocalizedMessage();
                    Log.e("CustomError", "---> " + ex.getLocalizedMessage());
                } catch (Exception ex){
                    msg = ex.getMessage();
                }
            }
        }.execute();
    }
}

```

```

    }
    return msg;
  }

  @Override
  protected void onPostExecute(String msg) {

    if( awsRecipientUsers.size() != 0 ){

      new AsyncTask<Void, Void, String>() {
        @Override
        protected String doInBackground(Void... params) {
          String msg = "";
          try {
            // For current date
            Calendar cur_cal = Calendar.getInstance();
            Date dt = cur_cal.getTime();
            SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-
dd'T'HH:mm:ss'Z'");
            dateFormat.setTimeZone(TimeZone.getTimeZone("en_US_POSIX"));

            //for chat room id
            UUID uuid = UUID.randomUUID();
            randomUUIDString = uuid.toString();

            ChatRoomManager chatRoomManager = new ChatRoomManager();
            // create hash set
            Set<String> endpointSet = new HashSet<String>();
            for(UserProfileDO userProfileDO : awsRecipientUsers){
              endpointSet.add(userProfileDO.getUserId());
            }
            chatRoomManager.saveNewChatRoom(dateFormat.format(dt),
credentialsProvider, groupName.getText().toString(), endpointSet,
randomUUIDString);

            } catch (AmazonServiceException ex) {
              msg = ex.getLocalizedMessage();
              Log.e("CustomError", "---> " + ex.getLocalizedMessage());
            }catch (Exception ex){
              msg = ex.getMessage();
            }
            return msg;
          }
        }

        @Override
        protected void onPostExecute(String msg) {
          Intent intent = new Intent(CreateChatRoomActivity.this,
ChatActivity.class);
          intent.putExtra("ID",randomUUIDString);
          intent.putExtra("Flag",false);
          startActivity(intent);
          finish();

        }
      }.execute();

    }
  }.execute();
}
}
}.execute();

```

```
    }else {  
        new AlertDialog.Builder(CreateChatRoomActivity.this)  
            .setTitle(R.string.error)  
            .setMessage(R.string.error_message)  
            .setPositiveButton(android.R.string.ok, null)  
            .show();  
    }  
}
```

When a user creates a new chat room in `AWSSampleMessenger`, the user can only add recipients who are:

- Present in the user's address book
- Already registered with the application
- Have registered with a phone number that matches the recipient's entry in the user's address book

`showChatRooms` in `DashboardActivity` is called to display the chat room list.

```
private void showChatRooms(){  
  
    if(awsChatRoomsList.size() != 0){  
        //change and set date format in ChatRoomDO  
        for ( ChatRoomDO chatRoom : awsChatRoomsList) {  
            String changeDate =  
chatRoomListAdapter.setDateWithNewFormat(chatRoom.getCreatedAt());  
            chatRoom.setCreatedAt(changeDate);  
        }  
  
        for ( ChatRoomDO chatRoom : awsChatRoomsList) {  
            chatRoomListAdapter.add(chatRoom);  
        }  
  
        chatRoomListAdapter.sort();  
        chatListView.setAdapter(chatRoomListAdapter);  
        chatRoomListAdapter.notifyDataSetChanged();  
  
        chatListView.setOnItemClickListener(new  
AdapterView.OnItemClickListener() {  
            @Override  
            public void onItemClick(AdapterView<?> parent, View view, final  
int position, long id){  
                //call conversation activity  
                Intent intent = new  
Intent(DashBoardActivity.this,ChatActivity.class);  
                intent.putExtra("ID", chatRoomListAdapter.getId(position));  
                intent.putExtra("Flag",false);  
                startActivity(intent);  
            }  
        });  
    }  
}
```

## User Data Storage: Amazon S3

`AWSSampleMessenger` uses Amazon S3 to implement the user data storage feature of your app.

## Uploading an Image From Code

To enable upload of image files for sharing in chat sessions, the `uploadWithData()` function uses `ImageSelectorUtils` to provide a path to the `userFileManager` `uploadContent` method and facilitates the secure transfer to the S3 bucket used by the app.

```
private void uploadWithData(Intent data) {

    final Uri uri = data.getData();
    //get file path from a Uri
    final String path =
ImageSelectorUtils.getFileFromUri(ChatActivity.this, uri);
    File resizeFile = null;
    File file = new File(path);

    Bitmap bitmap = decodeSampledBitmapFromUri(path, 200, 150);
    try {
        resizeFile = saveBitmap(bitmap, file.getName());
    } catch (IOException e) {
        e.printStackTrace();
    }

    //get an instance of User File Manager which uploads files from Amazon S3
    userFileManager.uploadContent(resizeFile,resizeFile.getName(), new
ContentProgressListener() {
        @Override
        public void onSuccess(final ContentItem contentItem) {

            CognitoCachingCredentialsProvider credentialsProvider = new
CognitoCachingCredentialsProvider(
                getApplicationContext(),
                AWSConfiguration.AMAZON_COGNITO_IDENTITY_POOL_ID, //
Identity Pool ID
                AWSConfiguration.AMAZON_S3_USER_FILES_BUCKET_REGION //
Region
            );

            final AmazonS3 s3 =
                new AmazonS3Client(credentialsProvider);

s3.setRegion(Region.getRegion(AWSConfiguration.AMAZON_S3_USER_FILES_BUCKET_REGION));

            final String identityId = AWSMobileClient.defaultMobileClient()
                .getIdentityManager()
                .getCredentialsProvider()
                .getCacheIdentityId();

            //full path of upload image
            final URL presignedUrl =
s3.generatePresignedUrl(AWSConfiguration.AMAZON_S3_USER_FILES_BUCKET,
S3_PREFIX_PRIVATE + identityId + "/" +contentItem.getFileFromUri(),
                new Date(new Date().getTime() + 60 * 60 * 1000));

            String[] urlParts = presignedUrl.toString().split("\\?");

            uploadedImagePath = urlParts[0];

            sendBtn.callOnClick();
        }
    });
}
```

```

        Log.d("URL", uploadedImagePath);
    }

    @Override
    public void onProgressUpdate(final String fileName, final boolean
isWaiting,
                                final long bytesCurrent, final long
bytesTotal) {
        Log.d("URL", fileName);
    }

    @Override
    public void onError(final String fileName, final Exception ex) {

        Log.e("CustomError", "---> " + ex.getLocalizedMessage());
    }
    });
}

```

## Push Notifications: Amazon SNS

AWSSampleMessenger uses Amazon SNS to implement the push notifications feature of your app.

The following code establishes identifiers for the send-and-receive endpoints needed to complete a push notification interaction between users. In this code the identifier is called `targetARN` and is in the form of an Amazon Resource Name (ARN).

### Establishing the User's Push Endpoint

In `LoginActivity`, the following function interacts with the `UserProfile` table to establish identifiers for the endpoint of the user's device.

```

new AsyncTask<Void, Void, String>() {
    @Override
    protected String doInBackground(Void... params) {
        String msg = "";
        try {
            // register device first to ensure we have a push endpoint.
            pushManager.registerDevice();
            isUser =
userProfileManager.isUserExist(cachingCredentialsProvider);

            Log.i("AWS", msg);

        } catch (AmazonServiceException ex) {
            msg = ex.getLocalizedMessage();
            Log.e("CustomError", "---> " + ex.getLocalizedMessage());
        }
        return msg;
    }

    @Override
    protected void onPostExecute(String msg) {
        if (isUser != null) {
            if (isUser.getItems().size() == 0) {

                //get phone number

```



```

        LayoutInflater li = LayoutInflater.from(LoginActivity.this);
        View promptsView = li.inflate(R.layout.prompt, null);
        AlertDialog.Builder alertDialogBuilder = new
AlertDialog.Builder(LoginActivity.this);
        alertDialogBuilder.setView(promptsView);
        final EditText userInput = (EditText)
promptsView.findViewById(R.id.editTextDialogUserInput);
        // set dialog message
        alertDialogBuilder
            .setCancelable(false)
            .setPositiveButton(R.string.ok,
                new DialogInterface.OnClickListener() {
                    public void onClick(final DialogInterface
dialog, int id) {

                        new AsyncTask<Void, Void, String>() {
                            @Override
                            protected String
doInBackground(Void... params) {
                                String msg = "";
                                try {
                                    // create hash set
                                    Set<String> endpointSet = new
HashSet<String>();
                                    endpointSet.add(pushManager.getEndpointArn());
                                    String ph =
userInput.getText().toString();
                                    //add user profile
information in UserProfile table
                                    userProfileManager.addUserProfile(endpointSet, ph, identityManager,
cachingCredentialsProvider);

                                } catch (AmazonServiceException
ex) {
                                    msg =
ex.getLocalizedMessage();
                                    Log.e("CustomError", "---> "
+ ex.getLocalizedMessage());
                                }
                                return msg;
                            }
                            @Override
                            protected void onPostExecute(String
msg) {
                                startActivity(new
Intent(LoginActivity.this, DashBoardActivity.class)
                                    .setFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP));
                                finish();
                                dialog.cancel();
                            }
                        }.execute();
                    }
                })
            .setNegativeButton(R.string.cancel,
                new DialogInterface.OnClickListener() {

```

```

        public void onClick(DialogInterface dialog,
int id) {
            dialog.cancel();
        }
    });

    // create alert dialog
    AlertDialog alertDialog = alertDialogBuilder.create();
    // show it
    alertDialog.show();
} else {

    final String phNo =
isUser.getItems().get(0).get("phone").getS();
    List<String> endpointArn =
isUser.getItems().get(0).get("pushTargetArn").getSS();
    final Set<String> endpointSet = new HashSet<String>();
    endpointSet.add(pushManager.getEndpointArn());
    for (int i = 0; i < endpointArn.size(); i++) {
        endpointSet.add(endpointArn.get(i));
    }

    new AsyncTask<Void, Void, String>() {
        @Override
        protected String doInBackground(Void... params) {
            String msg = "";
            try {
                //add user profile information in UserProfile
                table
                userProfileManager.addUserProfile(endpointSet,
phNo, identityManager, cachingCredentialsProvider);

                } catch (AmazonServiceException ex) {
                    msg = ex.getLocalizedMessage();
                    Log.e("CustomError", "---> " +
ex.getLocalizedMessage());
                }
                return msg;
            }

            @Override
            protected void onPostExecute(String msg) {
                Log.d(LOG_TAG, "Launching DashBoard Activity...");
                startActivity(new Intent(LoginActivity.this,
DashboardActivity.class)
                    .setFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP));
            }
        }.execute();
    }
}
}.execute();

```

## Establishing the User's Push Endpoint and Pushing

ChatActivity contains the `sendPush()` function, which interacts with the `UserProfile` table to establish identifiers for a recipient's endpoint when a push notification is sent. In the following code, the function then sets a `PublishRequest` object's attributes using that recipient `targetArn`, along

with `MessageStructure` and `Message`. That object is passed to the `AmazonSNSClient` `publish` method.

**Note**

Sending push notification from mobile end points is not a best practice. Instructions are included in the sample application for demonstration purposes.

```
private void sendPush(){

    new AsyncTask<Void, Void, String>() {
        @Override
        protected String doInBackground(Void... params) {
            String msg = "";
            try {

                CognitoCachingCredentialsProvider credentialsProvider = new
                CognitoCachingCredentialsProvider(
                    getApplicationContext(),
                    AWSConfiguration.AMAZON_COGNITO_IDENTITY_POOL_ID, //
                Identity Pool ID
                    AWSConfiguration.AMAZON_SNS_REGION // Region
                );

                AmazonSNSClient snsClient = new
                AmazonSNSClient(credentialsProvider.getCredentials());

                snsClient.setRegion(Region.getRegion(AWSConfiguration.AMAZON_SNS_REGION));
                for (UserProfileDO userData : awsUsersData) {
                    String currentUserID =
                credentialsProvider.getCachedIdentityId();
                    String userID = userData.getUserId();
                    if(!userID.equals(currentUserID)){
                        Set<String> targetArn = userData.getPushTargetArn();
                        // create an iterator
                        Iterator targetArnIterator = targetArn.iterator();
                        for( ;targetArnIterator.hasNext());{
                            PublishRequest publishRequest = new
                PublishRequest();
                            publishRequest.setMessageStructure("json");

                publishRequest.setTargetArn(targetArnIterator.next().toString());
                            String sender =
                identityManager.getCurrentIdentityProvider().getUserName();
                            String defaultMessage = String.format("\"default
                \": \"Sent By %s\", \", \", sender);
                            String gcmMessage = String.format("\"GCM\": \"{\\
                \\data\\\\\": {\\\\"message\\\\\": \\\\"Message sent by %s\\\\\", \\\\"chatRoomId\\\\\": \\
                \\%s\\\\\"} }\", \", \", sender, chatRoomId);
                            String apnsMessage = String.format("\"APNS\": \"{\\
                \\aps\\\\\": {\\\\"alert\\\\\": \\\\"Message sent by %s\\\\\", \\\\"chatRoomId\\\\\": \\
                \\%s\\\\\"} }\", \", \", sender, chatRoomId);
                            String apnsSANDBOXMessage =
                String.format("\"APNS_SANDBOX\": \"{\\\\"aps\\\\\": {\\\\"alert\\\\\": \\\\"Message
                sent by %s\\\\\", \\\\"chatRoomId\\\\\": \\\\"%s\\\\\"} }\" \", \", sender, chatRoomId);
                            String message = String.format("{\n" +
                defaultMessage + gcmMessage + apnsMessage + apnsSANDBOXMessage + "}");
                            publishRequest.setMessage(message);
                            try{
```

```

        PublishResult publishResult =
snsClient.publish(publishRequest);
        Log.i("AWS", publishResult.getMessageId());
    } catch (AmazonServiceException ex) {
        msg = ex.getLocalizedMessage();
        Log.e("CustomError", "---> " +
ex.getLocalizedMessage());
    }
    }
    }
    } catch (AmazonServiceException ex) {
        msg = ex.getLocalizedMessage();
        Log.e("CustomError", "---> " + ex.getLocalizedMessage());
    } catch (Exception ex) {
        msg = ex.getMessage();
    }
    return msg;
}

@Override
protected void onPostExecute(String msg) {
    loadChat();
    messageBox.setText("");
    uploadedImagePath = "";
}
}.execute();
}

```

## Receiving Push Notifications from Code

This section describes how the app receives and processes a chat notification both when the chat activity is open and in the background when the app is closed. In both cases, the app obtains the `chatRoomId` from the push data sent to the receiving device.

When the chat activity is open, the following code responds to an incoming push notification.

```

private final BroadcastReceiver notificationReceiver = new
BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {

        Bundle data =
intent.getBundleExtra(PushListenerService.INTENT_SNS_NOTIFICATION_DATA);
        chatRoomId = data.getString("chatRoomId");
        if(chatRoomId != null){loadChat();}
    }
};

```

When the chat activity is not open the following code responds to an incoming push notification.

```

private final BroadcastReceiver notificationReceiver = new
BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        final Bundle data =
intent.getBundleExtra(PushListenerService.INTENT_SNS_NOTIFICATION_DATA);

```

```
        new AlertDialog.Builder(DashboardActivity.this)
            .setTitle(R.string.push_demo_title)
            .setMessage(data.getString("message"))
            .setPositiveButton(android.R.string.ok, new
DialogInterface.OnClickListener() {
                @Override
                public void onClick(DialogInterface dialog, int which) {
                    //call conversation activity
                    Intent intent = new
Intent(DashboardActivity.this, ChatActivity.class);
                    intent.putExtra("ID", data.getString("chatRoomId"));
                    startActivity(intent);
                }
            })
            .show();
    }
};
```

## iOS Source Code

This section describes how the AWS back-end services configured for this iOS mobile app are integrated and invoked.

### Topics

- [User Sign-in: Amazon Cognito and Identity Providers \(p. 103\)](#)
- [NoSQL Database: Amazon DynamoDB and the AWSSampleMessenger Schema \(p. 104\)](#)
- [User Data Storage: Amazon S3 \(p. 109\)](#)
- [Push Notifications: Amazon SNS \(p. 110\)](#)

## User Sign-in: Amazon Cognito and Identity Providers

AWSSampleMessenger uses Amazon Cognito Identity to implement the sign-in feature of your app.

### Sign-in Required

AWSSampleMessenger user sign-in is configured so a user must provide credentials that are authenticated by an identity provider (Facebook Login or Google+ Sign-in, in this instance). When a user signs in, a three-way communication occurs between the mobile client, the identity provider, and Amazon Cognito. Successful authentication results in the user having an authenticated identity in the Amazon Cognito identity pool that is associated with your version of this app.

In configuring the User Sign-in feature, Mobile Hub provisions AWS Identity and Access Management (IAM) roles and policies regarding authenticated and unauthenticated access to the resources you have configured, such as Amazon DynamoDB tables. For AWSSampleMessenger, a policy allowing access is attached to the authenticated role. That authenticated role is attached to the Amazon Cognito identity pool for your app, allowing any signed-in user access to the back-end resources for the app.

### User Sign-in Components

#### Mobile Client

The Mobile Client bootstraps the app. It creates an identity manager to establish the user identity with Amazon Cognito. It also enables the features you selected for the project. For example, it handles the push notification device tokens, and indicates to the Amazon Mobile Analytics client when to

pause and resume metrics collection for the user session. Code for this component can be found in `MySampleApp/AmazonAws/AWSMobileClient.swift`.

### Identity Manager

The Identity Manager keeps track of the identity of the current app user by storing the user's Amazon Cognito identity pool ID. The Identity Manager assists with signing the user into the app if you have User Sign-in enabled in your project. It passes results of the sign-in operation back to the application's handler. Code for this component can be found in `AWSIdentityManager.swift`.

### Sign-in Manager

The Sign-in Manager is a single component that creates the sign-in identity providers and orchestrates sign-in call flows. The Sign-in Manager is responsible for keeping track of the most recent provider, refreshing credentials, and initializing sign-in buttons with the respective sign-in providers. In iOS, use `AWSIdentityManager`. Then pass the `AWSSignInProvider` object to its `loginWithSignInProvider` method.

## NoSQL Database: Amazon DynamoDB and the AWSSampleMessenger Schema

AWSSampleMessenger uses Amazon DynamoDB to implement the NoSQL database feature of your app.

### Table Indexes

Each table contains a primary index that consists of a primary key and a sort key, both of which organize data for most efficient query path. The `UserProfile` table is designed so that queries of `userId` and `phone` have the best performance. The `ChatRoom` and `Conversation` tables also have a secondary index called `ByCreationDate` to maximize query performance based on the chat room and the date when the chat room was created or the conversation happened.

### Data Relationships

The tables contain some attributes in common. These common attributes indicate to the app the relationship between items (equivalent to rows in a relational database). For instance, the recipients participating in a Chatroom can be enumerated for a given conversation by querying the `ChatRoom` table `recipients` attribute value based on the shared `chatRoomId` value.

### Security

The tables of this app are all *Protected*, which in Mobile Hub means the owner of an item in a table can read and modify an item, everyone else can read but not modify the item data. The owner, in this case, is the Amazon Cognito Identity ID connected to the instance of the app that created the item. This security is enforced in the following ways:

- When a user takes an action in the app that creates an item, the user's Amazon Cognito Identity ID is captured as the `userId` attribute value for the item.
- Amazon DynamoDB table permissions have been configured such that when a user takes an action that would modify the data of that item, the `userId` value is compared to the Amazon Cognito Identity ID of the active user. If these values match, then modifying the data is allowed.

### Schema

The following summarizes the table definitions of the AWSSampleMessenger schema.

Table Name: `UserProfile`

Table property	Type of value	Value
Security		Protected
Primary Key	string	<code>userId</code>
Sort Key	string	<code>phone</code>
Attribute	string	<code>name</code>
Attribute	string set	<code>pushTargetArn</code>

Table Name: `Chatroom`

Table property	Type of value	Value
Security		Protected
Primary Key	string	<code>userId</code>
Sort Key	string	<code>chatRoomId</code>
Attribute	string	<code>createdAt</code>
Attribute	string	<code>name</code>
Attribute	string set	<code>recipients</code>
Secondary Index Name		<code>ByCreationDate</code>
Secondary Index Primary Key	string	<code>chatRoomId</code>
Secondary Index Sort Key	string	<code>createdAt</code>

Table Name: `Conversation`

Table property	Type of value	Value
Security		Protected
Primary Key	string	<code>userId</code>
Sort Key	string	<code>conversationId</code>
Attribute	string	<code>createdAt</code>
Attribute	string	<code>chatRoomId</code>
Attribute	string	<code>imageUrlPath</code>
Attribute	string	<code>message</code>
Secondary Index Name		<code>ByCreationDate</code>
Secondary Index Primary Key	string	<code>chatRoomId</code>
Secondary Index Sort Key	string	<code>createdAt</code>

## How DynamoDB is Used

The following code in `CreateChatRoomVC` uses the `AWSSampleMessenger` schema to enable users to create chat rooms and initiate conversations with one or multiple contacts from the device's address book.

When a user creates a new chat room in `AWSSampleMessenger`, the user can only add recipients who are:

- Present in the user's address book
- Already registered with the application
- Have registered with a phone number that matches the recipient's entry in the user's address book

To verify whether the selected phone number is registered with the application, implement the `isUserRegistered` function.

```
func isUserRegistered(contact: CNContact) {
    if let number = contact.phoneNumbers.first?.value as? CNPhoneNumber {

        showBusyIndicator(true)

        userServices.getUserFromPhoneNo(number.stringValue).continueWithBlock({ (task)
        -> AnyObject? in
            self.showBusyIndicator(false)
            if let _userProfile = task.result as? UserProfile {

                print(_userProfile)
                dispatch_async(dispatch_get_main_queue(), {
                    self.recipeintsDataSource.append(contact)
                    self.tableView.reloadData()
                })
            }else{
                UIAlertController.showErrorAlertWithMessage("This phone
number is not registered")
            }
            return nil
        })
    }
}
```

`loadUsersWithPhoneList` retrieves an enumeration of selected recipients from the `UserProfile` table. Running this code enables you to load all contacts from the authenticated user's database.

```
func loadUsersWithPhoneList(phoneList: Array<String>)->AWSTask {
    let scanExpression = AWSDynamoDBScanExpression()
    var filters = Dictionary<String,String>()
    for index in 0...phoneList.count-1 {
        filters["val\(index)"] =
(phoneList[index].componentsSeparatedByCharactersInSet(NSCharacterSet.decimalDigitCharacter
)
    }
    let allKeys = Array(filters.keys)
    let keysExpression = allKeys.joinWithSeparator(",")
    scanExpression.filterExpression = "phone in \(keysExpression)"
    scanExpression.expressionAttributeValues = filters
    return dynamoDBObjectMapper!.scan(UserProfile.self,
expression:scanExpression).continueWithBlock { (task) -> AnyObject? in
```



```
        if task.error != nil || task.exception != nil {
            print(task.exception)
            return AWSTask(error: NSError(domain: "", code: -11, userInfo: [
                NSLocalizedDescriptionKey: "Users are not found!"
            ]))
        }
        if task.result != nil {
            print(task.result)
            let paginatedOutput:AWS DynamoDBPaginatedOutput = task.result as!
            AWS DynamoDBPaginatedOutput;
            for rect in paginatedOutput.items {
                print(rect)
            }
            return AWSTask(result: paginatedOutput.items)
        }
        return nil
    }
}
```

`saveNewChatRoom` inserts a new chat room in the `ChatRoom` table, which allows the user to create a new chat room, with one or more recipients obtained from the data produced by `loadUsersWithPhoneList`.

```
func saveNewChatRoom(chatRoomName:String?,userProfiles:[UserProfile])-
->AWSTask {
    let chatRoom = ChatRoom();

    chatRoom._chatRoomId = NSUUID().UUIDString
    chatRoom._createdAt = NSDate().formattedISO8601
    chatRoom._userId =
    AWSIdentityManager.defaultIdentityManager().identityId!
    chatRoom._name = chatRoomName
    chatRoom._recipients = Set<String>()
    for userProfile in userProfiles {
        chatRoom._recipients?.insert(userProfile._userId!)
    }
    //Save
    return dynamoDBObjectMapper!.save(chatRoom).continueWithBlock { (task) ->
    AnyObject? in
        if task.error != nil || task.exception != nil {
            print(task.exception)
            return AWSTask(error: NSError(domain: "", code: -11, userInfo: [
                NSLocalizedDescriptionKey: "Chat room is not created"
            ]))
        }
        if task.result != nil {
            return AWSTask(result: "Chat Room Created")
        }
        return nil
    }
}
```

Create the `CreateChatRoomVC` function.

```
@IBAction func createChatRoom(sender: UIButton) {

    if recipeintsDataSource.count > 0 {
```

```

        showBusyIndicator(true)

userServices.loadUsersWithPhoneList(getAllSelectedPhone()).continueWithBlock({ (task)
-> AnyObject? in
    if let _userProfiles = task.result as? [UserProfile] where
_userProfiles.count > 0 {
        let name = self.chatRoomNameTextField.text?.isEmpty == true ?
nil : self.chatRoomNameTextField.text
        return self.chatServices.saveNewChatRoom(name, userProfiles:
_userProfiles)
    }
    return nil
}).continueWithBlock({ (task) -> AnyObject? in

    if let result = task.result as? String {
        print(result)

        dispatch_async(dispatch_get_main_queue(), {

self.navigationController?.popViewControllerAnimated(true)

NSNotificationCenter.defaultCenter().postNotificationName("ReLoadChatRooms",
object: nil)
        })
    }
    self.showBusyIndicator(false)
    return nil
})
}
else {
    let alertController = UIAlertController(title: "Error", message:
"Pleas Add at least one recipient user", preferredStyle: .Alert)
    let doneAction = UIAlertAction(title: "Cancel", style: .Cancel,
handler: nil)
    alertController.addAction(doneAction)
    presentViewController(alertController, animated: true, completion:
nil)
}
}
}

```

## Display Chat Room

Implement the `loadUserChatRooms` function to load all associated chat rooms for a logged-in user.

```

func loadUserChatRooms()->AWSTask {
    let loggedInUserId =
AWSIdentityManager.defaultIdentityManager().identityId!
    let scanExpression = AWSDynamoDBScanExpression()
    scanExpression.filterExpression = "userId = :userId or
contains(recipients, :userId)"
    scanExpression.expressionAttributeValues = [":userId":loggedInUserId]
    return dynamoDBObjectMapper!.scan(ChatRoom.self,
expression:scanExpression).continueWithBlock { (task) -> AnyObject? in

        if let _result = task.result {
            print(_result)
            let paginatedOutput:AWSDynamoDBPaginatedOutput = _result as!
AWSDynamoDBPaginatedOutput;

```

```
        return AWSTask(result: paginatedOutput.items)
    }
    if let _error = task.error {
        print(_error)
    }
    if let _exception = task.exception {
        print(_exception)
    }
    return AWSTask(error: NSError(domain: "", code: -11, userInfo: [
        NSLocalizedDescriptionKey: "Recipient is not found"
    ]))
}
}
```

Now implement the `showChatRooms` function in `ChatTableViewController`.

```
@IBAction func showChatRooms(sender: UIRefreshControl) {
    if AWSIdentityManager.defaultIdentityManager().loggedIn == false {

        return
    }
    self.refreshControl?.beginRefreshing()
    chatServices.loadUserChatRooms().continueWithBlock { (task) -> AnyObject?
in
        sender.endRefreshing()
        if let _chatRooms = task.result as? Array<ChatRoom> {
            self.chatRoomDataSource?.removeAll()
            self.chatRoomDataSource = _chatRooms
            //Sort loaded chat rooms by creation date
            self.chatRoomDataSource?.sortInPlace({ (item1, item2) -> Bool in
                let date1 = NSDate().formattedISO8601Date(item1._createdAt!)
                let date2 = NSDate().formattedISO8601Date(item2._createdAt!)
                return date1.compare(date2) == .OrderedDescending
            })
        }
        self.tableView.reloadData()
        return nil
    }
}
```

## User Data Storage: Amazon S3

AWSSampleMessenger uses Amazon S3 to implement the user data storage feature of your app.

### Uploading an Image From Code

To enable upload of image files for sharing in chat sessions, the `uploadWithData()` function uses `UIImagePickerController` to pass the selected image as `NSData` to this function, and facilitate the secure transfer to the app's Amazon S3 bucket.

```
private func uploadWithData(data: NSData) {
    // convert current date into string
    let createdAt = NSString(format: "%@", NSDate()) as String
    //set image name with current date
    let imageName = "\(createdAt).png"
    //set upload destination folder in key
```

```

    let key = "private/
    \(AWSIdentityManager.defaultIdentityManager().identityId!)/\(imageName)"

    // create & initialize Conversation object
    let conversation = createConversation()
    conversation._imageUrlPath = imageName
    conversation._message = "IMAGE"
    //get An instance of `AWSLocalContent` that represents data to be
    uploaded.
    let localContent = userFileManager.localContentWithData(data, key: key)

    //start uploading from this function
    uploadLocalContent(localContent , conversation: conversation)
}

private func uploadLocalContent(localContent: AWSLocalContent ,
conversation:Conversation) {
    showUploadingStatusView(true)

    localContent.uploadWithPinOnCompletion(false, progressBlock: {[weak self]
(content: AWSLocalContent?, progress: NSProgress?) -> Void in
        // You can get uploading progress here ..
        }, completionHandler: {[weak self](content: AWSContent?, error:
NSError?) -> Void in
            guard let strongSelf = self else { return }
            strongSelf.showUploadingStatusView(false)
            if let error = error {
                print("Failed to upload an object. \(error)")
            } else {
                content?.getRemoteFileURLWithCompletionHandler({ (url, error)
in
                    // get full path of uploaded image
                    let imagePath =
url?.absoluteString.componentsSeparatedByString("?").first

                    // store into conversation object
                    conversation._imageUrlPath = imagePath

                    print(url?.absoluteString)
                    // send conversation object to Conversation Table
                    strongSelf.sendMessageToServer(conversation)

                })
            })
        })
}

```

## Push Notifications: Amazon SNS

AWSSampleMessenger uses Amazon SNS to implement the push notifications feature of your app.

The following code establishes identifiers for the send-and-receive endpoints that are needed to complete a push notification interaction between users. In this code, the identifier is called `targetARN` and is in the form of an Amazon Resource Name (ARN).

### Establishing the User's Push Endpoint

Implement the `getDeviceArn` function to get the device `targetArn`.

```
class func getDeviceArn() -> String? {
    let pushManager: AWSPushManager = AWSPushManager.defaultPushManager()
    if let _endpointARN = pushManager.endpointARN {
        return _endpointARN
    }else{
        pushManager.registerForPushNotifications()
    }
    return nil
}
```

After a successful log in, update the current user's `pushTargetArn` field in the `UserProfile` table.

## Sending Push Notifications From Code

`ConversationViewController` contains the `sendPush(conversation:Conversation)` function, which interacts with the `UserProfile` table to establish identifiers for a recipient's endpoint when a push notification is sent. In the following code, the function then sets an `AWSSNSPublishInput` object's attributes using the `targetArn` of the recipient, along with `MessageStructure` and `Message`. That object is passed to the `AWSSNS` method `publish()`.

- Set attributes for the `AWSSNSPublishInput.swift` class. Attributes include `targetArn` (recipient device ID), `messageStructure`, and `Message`.
- Pass the `AWSSNSPublishInput.swift` object to the `AWSSNS` method, `publish`.

### Note

Sending push notification from mobile end points is not a best practice. Instructions are included in the sample application for demonstration purposes.

```
func sendPush(conversation:Conversation) {
    let credentialsProvider =
    AWSCognitoCredentialsProvider(regionType:regionType(NSBundle.getRegionFromCredentialProvi
    identityPoolId:poolID)
    let configuration =
    AWSServiceConfiguration(region:regionType(NSBundle.getRegionFromPushManager()),
    credentialsProvider:credentialsProvider)
    AWSServiceManager.defaultServiceManager().defaultServiceConfiguration =
    configuration

    for userProfile in recipientUsers {
        //skip to current user
        if userProfile._userId ==
        AWSIdentityManager.defaultIdentityManager().identityId {
            continue
        }

        if let targetArns = userProfile._pushTargetArn {
            for deviceTargetArn in targetArns {
                do {
                    let sns = AWSSNS.defaultSNS()
                    let request = AWSSNSPublishInput()
                    request.messageStructure = "json"
                    let senderName =
                    AWSIdentityManager.defaultIdentityManager().userName

                    let devicePayload = ["default": "Message
                    sent by \(senderName!)", "APNS_SANDBOX": "{\"aps\":{\"alert\":
                    \"Message sent by \(senderName!)\", \"sound\": \"default\", \"badge\":
```



```
        }
        return nil
    })
}

func showMessageInConversation(chatRoom:ChatRoom , defaultMessage:String) {
    guard let _navigationController = self.window?.rootViewController as?
    UINavigationController else{
        showPushAlert(defaultMessage)
        return
    }
    guard let conversationVC = _navigationController.topViewController as?
    ConversationViewController where
    conversationVC.selectedChatRoom!._chatRoomId == chatRoom._chatRoomId else{
        showPushAlert(defaultMessage)
        return
    }
    conversationVC.selectedChatRoom = chatRoom
    conversationVC.loadRecipientsAndConversations(false)
    print(conversationVC)
}

func showPushAlert(defaultMessage:String) {
    dispatch_async(dispatch_get_main_queue(),{
        let alertController = UIAlertController(title: "Message", message:
        defaultMessage, preferredStyle: .Alert)
        let doneAction = UIAlertAction(title: "Cancel", style: .Cancel, handler:
        nil)
        alertController.addAction(doneAction)
        self.window?.rootViewController?.presentViewController(alertController,
        animated: true, completion: nil)
    })
}
```

# Document History for AWS Mobile Hub

The following table describes important changes to the documentation since the release of AWS Mobile Hub.

- **Latest documentation update:** August 19, 2016

Change	Description	Date Changed
AWS Mobile Hub Developer Guide Redesign	Site restructured around using the key mobile app backend features Mobile Hub facilitates. Most pages in the site are updated with additional information.	August 17, 2016
iOS and Android Push Notification Setup Documentation	The documentation for setting up iOS push notifications in the Apple Developer Member Center website and Android push notifications in the Google Developers Console website has been updated to provide more detail about the process for setting up these features outside the Mobile Hub console.	February 9, 2016
Facebook and Google Authentication Process Documentation	The documentation now has a section describing how to create a Google Developers Console project and create all client IDs Mobile Hub needs to enable Google Sign-In in both iOS and Android apps. For more information, see <a href="#">Setting Up Google</a>	January 26, 2016



Change	Description	Date Changed
	<a href="#">Authentication (p. 60)</a> . The documentation on creating a Facebook app ID has been updated to reflect changes in the Facebook Developer portal. For more information, see <a href="#">Setting Up Facebook Authentication (p. 57)</a> .	
IAM Managed Policies Added	Details about the managed policies required to view and modify configuration for any project with AWS Mobile Hub. For more information, see <a href="#">AWS Managed (Predefined) Policies for Mobile Hub Project Access (p. 117)</a> .	January 4, 2016
IAM Service Role for Mobile Hub Added	Details about the service policy and permissions for the <code>MobileHub_Service_Role</code> IAM role created by Mobile Hub to configure the features of each mobile app is documented. For more information, see <a href="#">Mobile Hub Service Role and Policies Used on Your Behalf (p. 118)</a> .	November 9, 2015
New Guide	This is the first release of the AWS Mobile Hub service. This is a beta release and is subject to change.	October 8, 2015

# AWS Mobile Hub Reference

---

The reference topics in this section provide more detailed information about how Mobile Hub works.

## Topics

- [AWS Identity and Access Management Usage in AWS Mobile Hub \(p. 116\)](#)
- [Amazon S3 Security Considerations for Mobile Hub Users \(p. 131\)](#)

## AWS Identity and Access Management Usage in AWS Mobile Hub

### **Note**

In depth understanding of IAM and AWS authentication and access controls is not required to build a mobile app using AWS Mobile Hub.

## Controlling Access to Your Mobile Hub Project

To learn how to grant permissions for configuration of your Mobile Hub project, see [Using AWS Managed Policies to Control Access to Mobile Hub Projects \(p. 117\)](#).

## Understanding Mobile Hub Permissions

To learn more about permissions you give Mobile Hub to configure AWS resources and services, see [Mobile Hub Service Role and Policies Used on Your Behalf \(p. 118\)](#)

## Understanding AWS Identity and Access Management

To learn about the details of IAM and AWS authentication and access controls, see [IAM Authentication and Access Control for Mobile Hub \(p. 126\)](#).

## Using AWS Managed Policies to Control Access to Mobile Hub Projects

This section describes how to control access to your projects using the **AWSMobileHub\_FullAccess** and **AWSMobileHub\_ReadOnly** AWS managed policies provided by Mobile Hub.

To understand how Mobile Hub uses IAM policies attached to the **MobileHub\_Service\_Role** to create and modify services on your behalf, see [Mobile Hub Service Role and Policies Used on Your Behalf](#) (p. 118).

To understand AWS Identity and Access Management (IAM) in more detail, see [IAM Authentication and Access Control for Mobile Hub](#) (p. 126) and [Overview of Access Permissions Management for Mobile Hub Projects](#) (p. 128).

### AWS Managed (Predefined) Policies for Mobile Hub Project Access

The AWS Identity and Access Management service controls user permissions for AWS services and resources. Specific permissions are required in order to view and modify configuration for any project with AWS Mobile Hub. These permissions have been grouped into the following managed policies, which you can attach to an IAM user, role, or group.

- **AWSMobileHub\_FullAccess**

This policy provides read and write access to AWS Mobile Hub projects. Users with this policy attached to their IAM user, role, or group are allowed to create new projects, modify configuration for existing projects, and delete projects and resources. This policy also includes all of the permissions that are allowed under the **AWSMobileHub\_ReadOnly** managed policy. After you sign in to the Mobile Hub console and create a project, you can use the following link to view this policy and the IAM identities that are attached to it.

[https://console.aws.amazon.com/iam/home?region=us-east-1#policies/arn:aws:iam::aws:policy/AWSMobileHub\\_FullAccess](https://console.aws.amazon.com/iam/home?region=us-east-1#policies/arn:aws:iam::aws:policy/AWSMobileHub_FullAccess)

- **AWSMobileHub\_ReadOnly**

This policy provides read-only access to AWS Mobile Hub projects. Users with this policy attached to their IAM user, role, or group are allowed to view project configuration and generate sample quick start app projects that can be downloaded and built on a developer's desktop (e.g., in Android Studio or Xcode). This policy does not allow modification to Mobile Hub project configuration, and it does not allow the user to enable the use of AWS Mobile Hub in an account where it has not already been enabled. After you sign in to the Mobile Hub console and create a project, you can use the following link to view this policy and the IAM identities that are attached to it.

[https://console.aws.amazon.com/iam/home?region=us-east-1#policies/arn:aws:iam::aws:policy/AWSMobileHub\\_ReadOnly](https://console.aws.amazon.com/iam/home?region=us-east-1#policies/arn:aws:iam::aws:policy/AWSMobileHub_ReadOnly)

### Viewing the Mobile Hub Console with Read-only Permissions

If your IAM user, role, or group has read-only permissions for use in an AWS Mobile Hub project, then the project information you see in the console will not reflect any changes made outside of Mobile Hub. For example, if you remove a Cloud Logic API in API Gateway, it may still be present in the Cloud Logic Functions list of your Mobile Hub project, until a user with **mobilehub:SynchronizeProject** permissions visits the console. Users who are granted console access through the **AWSMobileHub\_FullAccess** policy have those permissions. If you need additional permissions in Mobile Hub, please contact your administrator and request the Full Access policy.

## Attaching a Managed Policy to a User, Role, or Group

To use these managed policies, a user with administrative privileges must attach one of them to a user, role or group in the AWS Identity and Access Management console.

### To attach a managed policy

1. Choose the link for the managed policy you want to attach.
  - [https://console.aws.amazon.com/iam/home?region=us-east-1#policies/arn:aws:iam::aws:policy/AWSMobileHub\\_FullAccess](https://console.aws.amazon.com/iam/home?region=us-east-1#policies/arn:aws:iam::aws:policy/AWSMobileHub_FullAccess)
  - [https://console.aws.amazon.com/iam/home?region=us-east-1#policies/arn:aws:iam::aws:policy/AWSMobileHub\\_ReadOnly](https://console.aws.amazon.com/iam/home?region=us-east-1#policies/arn:aws:iam::aws:policy/AWSMobileHub_ReadOnly)
2. Choose **Attached Entities**.
3. Choose **Attach**.
4. Choose the users, roles, or groups you want to grant permissions.
5. Choose **Attach Policy**.

## Mobile Hub Service Role and Policies Used on Your Behalf

The following section describes the `MobileHub_Service_Role` IAM role that allows Mobile Hub to create and modify your AWS resources and services for the project you configure.

To understand how to grant and restrict permissions to your projects in the Mobile Hub console, see [Using AWS Managed Policies to Control Access to Mobile Hub Projects](#) (p. 117).

To understand AWS Identity and Access Management in more detail, see [IAM Authentication and Access Control for Mobile Hub](#) (p. 126) and [Overview of Access Permissions Management for Mobile Hub Projects](#) (p. 128).

### Topics

- [Source of Mobile Hub Service Role Permissions](#) (p. 118)
- [Trust Relationship](#) (p. 118)
- [Administrative Privileges](#) (p. 119)
- [Service Policy](#) (p. 119)

## Source of Mobile Hub Service Role Permissions

AWS Mobile Hub provides an integrated console experience in which you select mobile back-end features you can access from a mobile app. When you select and enable a feature, Mobile Hub configures multiple AWS services and resources on your behalf. Configuring AWS service or resource requires your permission to allow Mobile Hub to manage AWS services and resources for you. When you agree to the Mobile Hub console one-time request to manage AWS resources and services for you, you are giving Mobile Hub permissions that allow it to create a pre-defined IAM administrative service role, called `MobileHub_Service_Role`.

After this service role is created, you can see it at [https://console.aws.amazon.com/iam/home?region=us-east-1#roles/MobileHub\\_Service\\_Role](https://console.aws.amazon.com/iam/home?region=us-east-1#roles/MobileHub_Service_Role).

## Trust Relationship

In the IAM console at [https://console.aws.amazon.com/iam/home?region=us-east-1#roles/MobileHub\\_Service\\_Role](https://console.aws.amazon.com/iam/home?region=us-east-1#roles/MobileHub_Service_Role), there is a section for the trust relationship. The trust relationship dictates

which entities can assume this role and make use of its permissions. The trust relationship for this service role has an access control policy that looks like this:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": "mobilehub.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

This access control policy dictates that only AWS Mobile Hub (`mobilehub.amazonaws.com`) can assume this role. This policy should not be modified. No other user or system can assume this role and use its permissions.

## Administrative Privileges

By allowing Mobile Hub to create and assume the `MobileHub_Service_Role` role, you give Mobile Hub permissions to create additional roles as necessary to support the features enabled in your project. The `MobileHub_Service_Role` gives Mobile Hub permission to enable any service policies necessary on these additional roles for proper operation of the mobile app.

There are no limits on the number or scope of service policies or roles Mobile Hub may create. Actions taken by Mobile Hub in this regard are always in response to your actions in the Mobile Hub console. Roles or policies are never created without direct action from you, such as creating a Mobile Hub project or configuring an app feature.

## Revoking Privileges

To disallow Mobile Hub access to any users of your account, delete the `MobileHub_Service_Role` role. Make sure your users don't have permission to re-create the role, for example by having the `IAM:CreateRole` permission.

## Service Policy

The service policy states which operations an entity that assumes the `MobileHub_Service_Role` role can perform. If the role has been created, go to [https://console.aws.amazon.com/iam/home?region=us-east-1#roles/MobileHub\\_Service\\_Role](https://console.aws.amazon.com/iam/home?region=us-east-1#roles/MobileHub_Service_Role) to see the service policy used by AWS Mobile Hub. It looks like the following example:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "cloudformation:CreateUploadBucket",
        "cloudformation:ValidateTemplate",
        "cloudfront:CreateDistribution",
        "cloudfront>DeleteDistribution",
        "cloudfront:GetDistribution",
        "cloudfront:GetDistributionConfig",

```

```
"cloudfront:UpdateDistribution",
"cognito-identity:CreateIdentityPool",
"cognito-identity:UpdateIdentityPool",
"cognito-identity>DeleteIdentityPool",
"cognito-identity:SetIdentityPoolRoles",
"cognito-idp:CreateUserPool",
"dynamodb:CreateTable",
"dynamodb>DeleteTable",
"dynamodb:DescribeTable",
"dynamodb:UpdateTable",
"iam:AddClientIDToOpenIDConnectProvider",
"iam:CreateOpenIDConnectProvider",
"iam:GetOpenIDConnectProvider",
"iam:ListOpenIDConnectProviders",
"iam:CreateSAMLProvider",
"iam:GetSAMLProvider",
"iam:ListSAMLProvider",
"iam:UpdateSAMLProvider",
"lambda:CreateFunction",
"lambda>DeleteFunction",
"lambda:GetFunction",
"mobileanalytics:CreateApp",
"mobileanalytics>DeleteApp",
"sns:CreateTopic",
"sns>DeleteTopic",
"ec2:DescribeSecurityGroups",
"ec2:DescribeSubnets",
"ec2:DescribeVpcs"
],
"Resource": [
  "*"
]
},
{
  "Effect": "Allow",
  "Action": [
    "sns:CreatePlatformApplication",
    "sns>DeletePlatformApplication",
    "sns:GetPlatformApplicationAttributes",
    "sns:SetPlatformApplicationAttributes"
  ],
  "Resource": [
    "arn:aws:sns:*:*:app/*_MOBILEHUB_*"
  ]
},
{
  "Effect": "Allow",
  "Action": [
    "s3:CreateBucket",
    "s3>DeleteBucket",
    "s3>DeleteBucketPolicy",
    "s3:ListBucket",
    "s3:ListBucketVersions",
    "s3:GetBucketLocation",
    "s3:GetBucketVersioning",
    "s3:PutBucketVersioning"
  ],
  "Resource": [
    "arn:aws:s3::*-userfiles-mobilehub-*",
```

```
    "arn:aws:s3:::*-contentdelivery-mobilehub-*",
    "arn:aws:s3:::*-deployments-mobilehub-*"
  ]
},
{
  "Effect": "Allow",
  "Action": [
    "s3:DeleteObject",
    "s3:DeleteVersion",
    "s3:DeleteObjectVersion",
    "s3:GetObject",
    "s3:GetObjectVersion",
    "s3:PutObject",
    "s3:PutObjectAcl"
  ],
  "Resource": [
    "arn:aws:s3:::*-userfiles-mobilehub-*/**",
    "arn:aws:s3:::*-contentdelivery-mobilehub-*/**",
    "arn:aws:s3:::*-deployments-mobilehub-*/**"
  ]
},
{
  "Effect": "Allow",
  "Action": [
    "lambda:AddPermission",
    "lambda:CreateAlias",
    "lambda>DeleteAlias",
    "lambda:UpdateAlias",
    "lambda:GetFunctionConfiguration",
    "lambda:GetPolicy",
    "lambda:RemovePermission",
    "lambda:UpdateFunctionCode",
    "lambda:UpdateFunctionConfiguration"
  ],
  "Resource": [
    "arn:aws:lambda:*:*:function:*-mobilehub-*"
  ]
},
{
  "Effect": "Allow",
  "Action": [
    "iam:CreateRole",
    "iam>DeleteRole",
    "iam>DeleteRolePolicy",
    "iam:GetRole",
    "iam:GetRolePolicy",
    "iam:ListRolePolicies",
    "iam:PassRole",
    "iam:PutRolePolicy",
    "iam:UpdateAssumeRolePolicy",
    "iam:AttachRolePolicy",
    "iam:DetachRolePolicy"
  ],
  "Resource": [
    "arn:aws:iam::*:role/*_unauth_MOBILEHUB_*",
    "arn:aws:iam::*:role/*_auth_MOBILEHUB_*",
    "arn:aws:iam::*:role/*_consolepush_MOBILEHUB_*",
    "arn:aws:iam::*:role/*_lambdaexecutionrole_MOBILEHUB_*",
    "arn:aws:iam::*:role/*_smsverification_MOBILEHUB_*",
```

```
        "arn:aws:iam::*:role/MOBILEHUB-*--lambdaexecution*",
        "arn:aws:iam::*:role/MobileHub_Service_Role"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents"
    ],
    "Resource": [
        "arn:aws:logs::*:log-group:/aws/mobilehub/*:log-stream:*"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "iam:ListAttachedRolePolicies"
    ],
    "Resource": [
        "arn:aws:iam::*:role/MobileHub_Service_Role"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "cloudformation:CreateStack",
        "cloudformation>DeleteStack",
        "cloudformation:DescribeStacks",
        "cloudformation:DescribeStackEvents",
        "cloudformation:DescribeStackResource",
        "cloudformation:GetTemplate",
        "cloudformation:ListStackResources",
        "cloudformation:UpdateStack"
    ],
    "Resource": [
        "arn:aws:cloudformation::*:stack/MOBILEHUB-*"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "apigateway:DELETE",
        "apigateway:GET",
        "apigateway:HEAD",
        "apigateway:OPTIONS",
        "apigateway:PATCH",
        "apigateway:POST",
        "apigateway:PUT"
    ],
    "Resource": [
        "arn:aws:apigateway::*/restapis*"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "cognito-idp>DeleteUserPool",
```



```
        "cognito-idp:DescribeUserPool",
        "cognito-idp:CreateUserPoolClient",
        "cognito-idp:DescribeUserPoolClient",
        "cognito-idp>DeleteUserPoolClient"
    ],
    "Resource": [
        "arn:aws:cognito-idp:*:*:userpool/*"
    ]
}
]
```

All of these permissions pertain to resources Mobile Hub creates on your behalf. You can see these resources by choosing **Resources** in the left navigation panel of the Mobile Hub console.

## AWS Identity and Access Management

These are the items in the service policy for the Mobile Hub service role defining IAM permissions.

```
"iam:CreateRole",
"iam>DeleteRole",
"iam>DeleteRolePolicy",
"iam:GetRole",
"iam:ListRolePolicies",
"iam:PassRole",
"iam:PutRolePolicy",
"iam:UpdateAssumeRolePolicy",
"iam:AttachRolePolicy",
"iam:DetachRolePolicy",
"iam:CreateSAMLProvider",
"iam:GetSAMLProvider",
"iam:ListSAMLProvider",
"iam:UpdateSAMLProvider"
```

Mobile Hub creates one or more IAM roles to use with your mobile app project, depending on the configuration options you choose for each feature. By default, IAM creates an unauthenticated app user role to allow users of your app to get temporary permissions to perform various operations with other services you've enabled. For example, you need this role when your app calls an AWS Lambda function in the Cloud Logic feature.

If you enable the Cloud Logic feature, Mobile Hub also creates an AWS Lambda execution role. This role provides your AWS Lambda functions the permissions they need to carry out their tasks; for example, writing debug logs to Amazon CloudWatch.

If you enable the User Sign-in feature, Mobile Hub creates an authenticated app user role. This authenticated app user role is used by the users of your app when they sign in using a sign-in provider such as Facebook or Google+. If you select the **Sign-in is required** option in User Sign-in, the unauthenticated app user role is removed. All access to your resources from the app then require use of the authenticated role.

In addition, if you select Google as a sign-in provider, Mobile Hub needs access to the following Open ID Connect Provider APIs from IAM:

```
"iam:AddClientIDToOpenIDConnectProvider",
"iam:CreateOpenIDConnectProvider",
"iam:GetOpenIDConnectProvider",
"iam:ListOpenIDConnectProviders",
```

These permissions allow the service to create an Open ID Connect Provider for Google if it does not already exist, and add ClientIDs to that provider.

## Amazon API Gateway

These are the items in the service policy for the Mobile Hub service role defining API Gateway permissions.

```
"apigateway:DELETE" ,  
"apigateway:GET" ,  
"apigateway:PATCH" ,  
"apigateway:POST" ,  
"apigateway:PUT" ,  
"apigateway:HEAD" ,  
"apigateway:OPTIONS"
```

These policies enable Mobile Hub to configure REST APIs for mobile back-ends.

## Amazon Cognito

These are the items in the service policy for the Mobile Hub service role defining Amazon Cognito permissions.

```
"cognito-identity:CreateIdentityPool" ,  
"cognito-identity:UpdateIdentityPool" ,  
"cognito-identity>DeleteIdentityPool" ,  
"cognito-identity:SetIdentityPoolRoles" ,  
"cognito-idp:CreateUserPool" ,  
"cognito-idp>DeleteUserPool" ,  
"cognito-idp:DescribeUserPool" ,  
"cognito-idp:CreateUserPoolClient" ,  
"cognito-idp:DescribeUserPoolClient" ,  
"cognito-idp>DeleteUserPoolClient" "
```

Amazon Cognito provides temporary credentials that give app users access to your AWS resources. By default Mobile Hub creates an Amazon Cognito identity pool to provide a scope or namespace for user identities. If you enable the User Sign-in feature and configure a sign-in provider, such as Facebook or Google+, Mobile Hub updates the identity pool to support that feature in your app.

## Amazon Elastic Compute Cloud

These are the items in the service policy for the Mobile Hub service role defining Amazon EC2 permissions.

```
"ec2:DescribeSecurityGroups" ,  
"ec2:DescribeSubnets" ,  
"ec2:DescribeVpcs"
```

## Amazon Mobile Analytics

These are the items in the service policy for the Mobile Hub service role defining Mobile Analytics permissions.

```
"mobileanalytics:CreateApp" ,  
"mobileanalytics>DeleteApp" ,
```

When you enable the App Analytics feature in Mobile Hub, it creates an App ID for your app in Amazon Mobile Analytics. This App ID can be removed if you delete the project.

## Amazon Simple Notification Service

These are the items in the service policy for the Mobile Hub service role defining Amazon SNS permissions.

```
"sns:CreateTopic",  
"sns>DeleteTopic",  
"sns:CreatePlatformApplication",  
"sns>DeletePlatformApplication",  
"sns:GetPlatformApplicationAttributes",  
"sns:SetPlatformApplicationAttributes",
```

When you enable the Push Notifications feature, Mobile Hub creates an Amazon SNS platform application for each push platform you configure. It also creates a default Amazon SNS topic you can use to push messages to all users of your app. The topic and platform application are deleted if you delete the associated Mobile Hub project.

## Amazon Simple Storage Service

These are the items in the service policy for the Mobile Hub service role defining Amazon S3 permissions.

```
"s3:CreateBucket",  
"s3>DeleteBucket",  
"s3>DeleteBucketPolicy",  
"s3:ListBucket",  
"s3:ListBucketVersions",  
"s3>DeleteObject",  
"s3>DeleteVersion",  
"s3:PutObject",  
"s3:PutObjectAcl",  
"s3:GetBucketLocation",  
"s3:GetObject",  
"s3:GetObjectVersion",
```

App Content Delivery and User Data Storage features both use Amazon Simple Storage Service. When you enable one of these features, Mobile Hub creates an Amazon S3 bucket on your behalf. Mobile Hub also puts example files and folders in the bucket so you can demonstrate your app downloading and navigating between folders. Some of these permissions are required to set up your Amazon S3 bucket for use with Amazon CloudFront if you enable the App Content Delivery feature and select Multi-Region CDN. Other policies enable storage capabilities needed by mobile back-end features that use multiple AWS services.

## Amazon CloudFront

These are the items in the service policy for the Mobile Hub service role defining CloudFront permissions.

```
"cloudfront:CreateDistribution",  
"cloudfront>DeleteDistribution",  
"cloudfront:GetDistribution",  
"cloudfront:GetDistributionConfig",  
"cloudfront:UpdateDistribution",
```

If you enable the App Content Delivery feature and configure it for Multi-Region CDN, Mobile Hub creates a CloudFront distribution with your Amazon S3 bucket set as the origin.

## AWS CloudFormation

These are the items in the service policy for the Mobile Hub service role defining AWS CloudFormation permissions.

```
"cloudformation:CreateUploadBucket",  
"cloudformation:ValidateTemplate",  
"cloudformation:CreateStack",  
"cloudformation:ListStackResources",  
"cloudformation>DeleteStack",  
"cloudformation:DescribeStacks",  
"cloudformation:DescribeStackEvents",  
"cloudformation:DescribeStackResource",  
"cloudformation:GetTemplate",  
"cloudformation:UpdateStack"
```

These policies allow Mobile Hub to dynamically provision and configure back-end stacks to support your mobile app's requirements.

## AWS Lambda

These are the items in the service policy for the Mobile Hub service role defining Lambda permissions.

```
"lambda:CreateFunction",  
"lambda>DeleteFunction",  
"lambda:GetFunction",  
"lambda:CreateAlias",  
"lambda>DeleteAlias",  
"lambda:GetFunctionConfiguration",  
"lambda:GetPolicy",  
"lambda:UpdateFunctionCode",  
"lambda:UpdateAlias",  
"lambda:UpdateFunctionConfiguration"
```

If you enable the Cloud Logic feature, Mobile Hub creates an example Lambda function. You can use this function to demonstrate invocation of a Lambda function from your app.

# IAM Authentication and Access Control for Mobile Hub

In depth understanding of AWS authentication and access controls is not required to build a mobile app using AWS Mobile Hub.

Mobile Hub uses AWS credentials and permissions policies in two ways:

- [Using AWS Managed Policies to Control Access to Mobile Hub Projects \(p. 117\)](#).
- Providing [Mobile Hub Service Role and Policies Used on Your Behalf \(p. 118\)](#) to create and configure the back-end features you select for your mobile app.

The following sections provide details on how IAM works, how you can use IAM to securely control access to your projects, and what IAM roles and policies Mobile Hub configures on your behalf.

Topics

- [Authentication](#) (p. 127)
- [Access Control](#) (p. 128)

## Authentication

AWS resources and services can only be viewed, created or modified with the correct authentication using AWS credentials (which must also be granted [access permissions](#) (p. 128) to those resources and services). You can access AWS as any of the following types of identities:

- **AWS account root user**

When you sign up for AWS, you provide an email address and password that is associated with your AWS account. These are your root credentials and they provide complete access to all of your AWS resources.

**Important**

For security reasons, we recommend that you use the root credentials only to create an administrator user, which is an IAM user with full permissions to your AWS account. Then, you can use this administrator user to create other IAM users and roles with limited permissions. For more information, see [IAM Best Practices](#) and [Creating an Admin User and Group](#) in the *IAM User Guide*.

- **IAM user**

An [IAM user](#) is simply an identity within your AWS account that has specific custom permissions (for example, read-only permissions to access your Mobile Hub project). You can use an IAM user name and password to sign in to secure AWS webpages like the [AWS Management Console](#), [AWS Discussion Forums](#), or the [AWS Support Center](#).

In addition to a user name and password, you can also generate [access keys](#) for each user. You can use these keys when you access AWS services programmatically, either through [one of the several SDKs](#) or by using the [AWS Command Line Interface \(CLI\)](#). The SDK and CLI tools use the access keys to cryptographically sign your request. If you don't use the AWS tools, you must sign the request yourself.

- **IAM role**

An [IAM role](#) is another IAM identity you can create in your account that has specific permissions. It is similar to an IAM user, but it is not associated with a specific person. An IAM role enables you to obtain temporary access keys that can be used to access AWS services and resources. IAM roles with temporary credentials are useful in the following situations:

- **Federated user access**

Instead of creating an IAM user, you can use preexisting user identities from your enterprise user directory or a web identity provider. These are known as federated users. AWS assigns a role to a federated user when access is requested through an [identity provider](#). For more information about federated users, see [Federated Users and Roles](#) in the *IAM User Guide*.

- **Cross-account access**

You can use an IAM role in your account to grant another AWS account permissions to access your account's resources. For an example, see [Tutorial: Delegate Access Across AWS Accounts Using IAM Roles](#) in the *IAM User Guide*.

- **AWS service access**

You can use an IAM role in your account to grant an AWS service permissions to access your account's resources. For example, you can create a role that allows Amazon Redshift to access an Amazon S3 bucket on your behalf and then load data stored in the bucket into an Amazon Redshift cluster. For more information, see [Creating a Role to Delegate Permissions to an AWS Service](#) in the *IAM User Guide*.

- **Applications running on Amazon EC2**

Instead of storing access keys within the EC2 instance for use by applications running on the instance and making AWS API requests, you can use an IAM role to manage temporary credentials for these applications. To assign an AWS role to an EC2 instance and make it available to all of its applications, you can create an instance profile that is attached to the instance. An instance profile contains the role and enables programs running on the EC2 instance to get temporary credentials. For more information, see [Using Roles for Applications on Amazon EC2](#) in the *IAM User Guide*.

## Access Control

You can have valid credentials to authenticate your requests, but unless you have permissions you cannot access or modify a Mobile Hub project. The same is true for Mobile Hub when it creates and configures services and resources you have configured for your project.

The following sections describe how to manage permissions and understand those that are being managed on your behalf by Mobile Hub.

- [Using AWS Managed Policies to Control Access to Mobile Hub Projects](#) (p. 117)
- [Mobile Hub Service Role and Policies Used on Your Behalf](#) (p. 118)

## Overview of Access Permissions Management for Mobile Hub Projects

In depth understanding of AWS authentication and access controls is not required to build a mobile app using AWS Mobile Hub.

Every AWS resource is owned by an AWS account, and permissions to create or access the resources are governed by permissions policies. This includes:

- Policies for [Using AWS Managed Policies to Control Access to Mobile Hub Projects](#) (p. 117).
- [AWS Mobile Hub Service Role and Policies](#) (p. 118) to create and configure the back-end features you select for your mobile app.

An account administrator can attach permissions policies to IAM identities (that is, users, groups, and roles), and some services (such as AWS Lambda) also support attaching permissions policies to resources.

### Note

An *account administrator* (or administrator user) is a user with administrator privileges. For more information, see [IAM Best Practices](#) in the *IAM User Guide*.

When granting permissions, you decide who is getting the permissions, the resources they get permissions for, and the specific actions that you want to allow on those resources.

### Topics

- [Understanding Resource Ownership for AWS Mobile Hub](#) (p. 129)
- [Managing Access to Resources](#) (p. 129)
- [Specifying Policy Elements: Actions, Effects, Resources, and Principals](#) (p. 130)

## Understanding Resource Ownership for AWS Mobile Hub

The primary resource of a Mobile Hub project is the project itself. In first use of the Mobile Hub console, you allow Mobile Hub to manage permissions and access the project resource for you. A resource owner is the AWS account that created a resource. That is, the resource owner is the AWS account of the principal entity (the root account, an IAM user, or an IAM role) that authenticates the request that creates the resource. The following examples illustrate how this works:

- If you use the root account credentials of your AWS account to create an AWS Mobile Hub project, your AWS account is the owner of the resources associated with that project.
- If you create an IAM user in your AWS account and grant permissions to create Mobile Hub projects to that user, the user can also create projects. However, your AWS account, to which the user belongs, owns the resources associated with the project.
- If you create an IAM role in your AWS account with permissions to create AWS Mobile Hub projects, anyone who can assume the role can create, edit, or delete projects. Your AWS account, to which the role belongs, owns the resources associated with that project.

## Managing Access to Resources

A *permissions policy* describes who has access to what. The following section explains the available options for creating permissions policies.

### Note

This section discusses using IAM in the context of AWS Mobile Hub. It doesn't provide detailed information about the IAM service. For complete IAM documentation, see [What Is IAM?](#) in the *IAM User Guide*. For information about IAM policy syntax and descriptions, see [AWS Identity and Access Management Policy Reference](#) in the *IAM User Guide*.

Policies attached to an IAM identity are referred to as identity-based policies (IAM policies) and policies attached to a resource are referred to as resource-based policies.

### Topics

- [Identity-Based Policies \(IAM Policies\)](#) (p. 129)
- [Resource-Based Policies](#) (p. 130)

## Identity-Based Policies (IAM Policies)

You can attach policies to IAM identities. For example, you can do the following:

- **Attach a permissions policy to a user or a group in your account** – An account administrator can use a permissions policy that is associated with a particular user to grant permissions for that user to view or modify an AWS Mobile Hub project.
- **Attach a permissions policy to a role (grant cross-account permissions)** – You can attach an identity-based permissions policy to an IAM role to grant cross-account permissions. For example, when you first enter Mobile Hub and agree, as account principal, to grant it permissions to provision and configure your project, you are granting the AWS managed `MobileHub_Service_Role` role cross-account permissions. An AWS managed policy, `AWSMobileHub_ServiceUseOnly`, is attached to that role in the context of your Mobile Hub project. The role has a trust policy that allows Mobile Hub to act as account principal with the ability to grant permissions for services and resources used by your project.

For more information about using IAM to delegate permissions, see [Access Management](#) in the *IAM User Guide*.

As an example of using an identity-based policy, the following policy grants permissions to a user to create an Amazon S3 bucket. A user with these permissions can create a storage location using the Amazon S3 service.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "s3:CreateBucket*",
      "Resource": "*"
    }
  ]
}
```

For more information about using identity-based policies with Mobile Hub , see [Using AWS Managed Policies to Control Access to Mobile Hub Projects \(p. 117\)](#) and [Mobile Hub Service Role and Policies Used on Your Behalf \(p. 118\)](#).

For more information about users, groups, roles, and permissions, see [Identities \(Users, Groups, and Roles\)](#) in the *IAM User Guide*.

## Resource-Based Policies

Other services, such as Amazon S3, also support resource-based permissions policies. For example, you can attach a policy to an Amazon S3 bucket to manage access permissions to that bucket.

## Specifying Policy Elements: Actions, Effects, Resources, and Principals

Each service that is configured by Mobile Hub defines a set of API operations. To grant Mobile Hub permissions for these API operations, a set of actions is specified in an AWS managed policy. Performing an API operation can require permissions for more than one action.

The following are the basic policy elements:

- **Resource** – In a policy, you use an Amazon Resource Name (ARN) to identify the resource to which the policy applies.
- **Action** – You use action keywords to identify resource operations that you want to allow or deny. For example, the `s3:Createbucket` permission allows Mobile Hub to perform the Amazon S3 `CreateBucket` operation.
- **Effect** – You specify the effect when the user requests the specific action—this can be either allow or deny. If you don't explicitly grant access to (allow) a resource, access is implicitly denied. You can also explicitly deny access to a resource, which you might do to make sure that a user cannot access it, even if a different policy grants access.
- **Principal** – In identity-based policies (IAM policies), the user that the policy is attached to is the implicit principal. For resource-based policies, you specify the user, account, service, or other entity that you want to receive permissions (applies to resource-based policies only).



# Amazon S3 Security Considerations for Mobile Hub Users

In Mobile Hub, if you use the User Data Storage feature, it creates an Amazon S3 bucket in your account. This topic describes the key Amazon S3 security-related features that you might want to use for this bucket.

## Object Lifecycle Management

You can use object lifecycle management to have Amazon S3 take actions on files (also referred to in Amazon S3 as *objects*) in a bucket based on specific criteria. For example, after a specific amount of time since a mobile app user uploaded a file to the bucket, you might want to permanently delete that file or move it to Amazon Glacier. You might want to do this to reduce the amount of data in files that other mobile app users can potentially access. You might also want to manage your costs by deleting or archiving files that you know you or mobile app users no longer need.

For more information, see [Object Lifecycle Management](#) in the *Amazon Simple Storage Service Developer Guide*.

## Object Encryption

Object encryption helps increase the protection of the data in files while they are traveling to and from a bucket as well as while they are in a bucket. You can use Amazon S3 to encrypt the files, or you can encrypt the files yourself. Files can be encrypted with an Amazon S3-managed encryption key, a key managed by AWS Key Management Service (AWS KMS), or your own key.

For more information, see the [Protecting Data Using Encryption](#) section in the *Amazon Simple Storage Service Developer Guide*.

## Object Versioning

Object versioning helps you recover data in files more easily after unintended mobile app user actions and mobile app failures. Versioning enables you to store multiple states of the same file in a bucket. You can uniquely access each version by its related file name and version ID. To help manage your costs, you can delete or archive older versions that you no longer need, or you can suspend versioning.

For more information, see the [Using Versioning](#) section in the *Amazon Simple Storage Service Developer Guide*.

## Bucket Logging

Bucket logging helps you learn more about your app users, helps you meet your organization's audit requirements, and helps you understand your Amazon S3 costs. Each access log record provides details about a single access request, such as the requester, bucket name, request time, request action, response status, and error code, if any. You can store logs in the same bucket or in a different one. To help manage your costs, you can delete logs that you no longer need, or you can suspend logging.

For more information, see [Managing Bucket Logging](#) in the *Amazon Simple Storage Service Console User Guide*.