AWS Mobile

Developer Guide

Table of Contents

What is AWS Mobile?

Cloud enable your app in minutes

AWS Mobile gives you the tools to rapidly configure and integrate the cloud backend your mobile app needs.

Android and iOS

Get Started (p. 2) using the AWS Mobile SDK.

Supported mobile app features include:

Analytics (p. 13) - User Sign-in (p. 20) - Push Notification (p. 44) - NoSQL Database (p. 54) - User File Storage (p. 66) - Cloud Logic - Messaging - Conversational Bots (p. 84) - Hosting and Streaming

AWS Mobile Hub lets you configure powerful backend services using a console.

Web and React Native

Get started with Web (p. 366) or React Native (p. 397) using the AWS Mobile CLI.

Supported mobile app features include:

Web - Analytics (p. 370) - User Sign-in (p. 371) - NoSQL Database (p. 372) - User File Storage (p. 377) - Cloud Logic (p. 379)

React Native Analytics (p. 399) - User Sign-in (p. 401) - NoSQL Database (p. 402) - User File Storage (p. 406) - Cloud Logic (p. 408)

Behind the scenes AWS Mobile Hub lets you configure powerful backend services from the command line with no AWS expertise needed.

Learn how its easy integrate your services by trying an AWS Mobile tutorial (p. 90).

AWS Mobile for Android and iOS

The AWS Mobile SDKs for Android and iOS, in combination with the AWS Mobile Hub, allow you to quickly and easily integrate robust cloud backends into your existing mobile apps. No AWS expertise is required to configure and begin to use features like user sign-in, database, push notifications and more.

Topics

Get Started

Overview

The AWS Mobile Android and iOS SDKs help you build high quality mobile apps quickly and easily. They provide easy access to a range of AWS services, including Amazon Cognito, AWS Lambda, Amazon S3, Amazon Kinesis, Amazon DynamoDB, Amazon Pinpoint and many more.

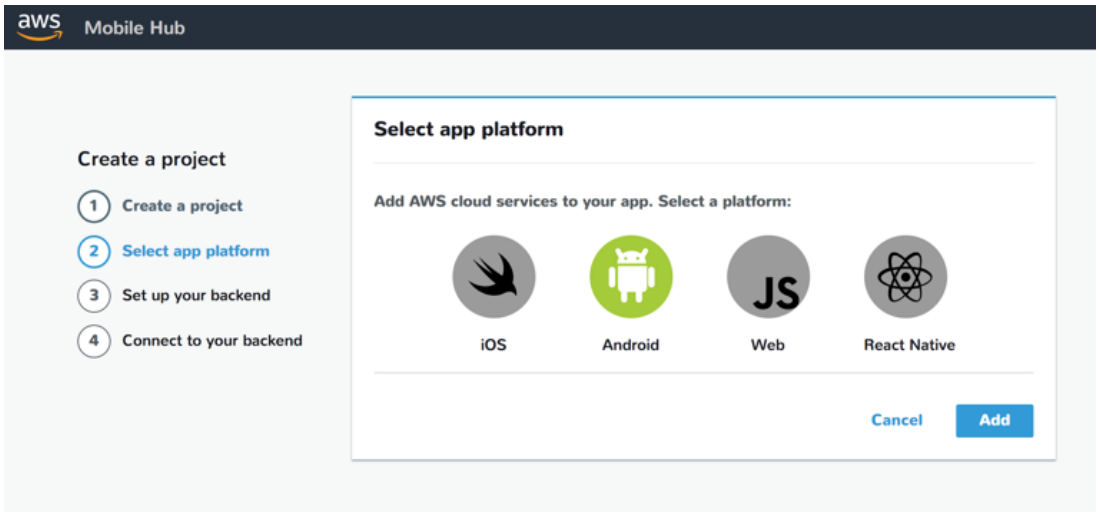
Set Up Your Backend

1. Sign up for the AWS Free Tier.
2. Create a Mobile Hub project by signing into the console. The Mobile Hub console provides a single location for managing and monitoring your app's cloud resources.

   To integrate existing AWS resources using the SDK directly, without Mobile Hub, see Setup Options for Android (p. 133) or Setup Options for iOS (p. 138).
3. Name your project, check the box to allow Mobile Hub to administer resources for you and then choose **Add**.

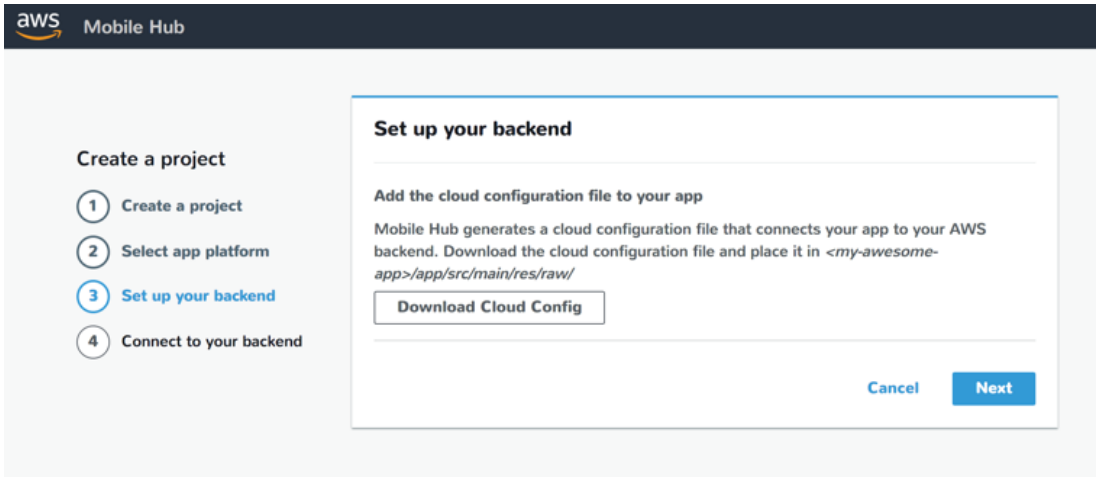
Android - Java

1. Choose **Android** as your platform and then choose Next.



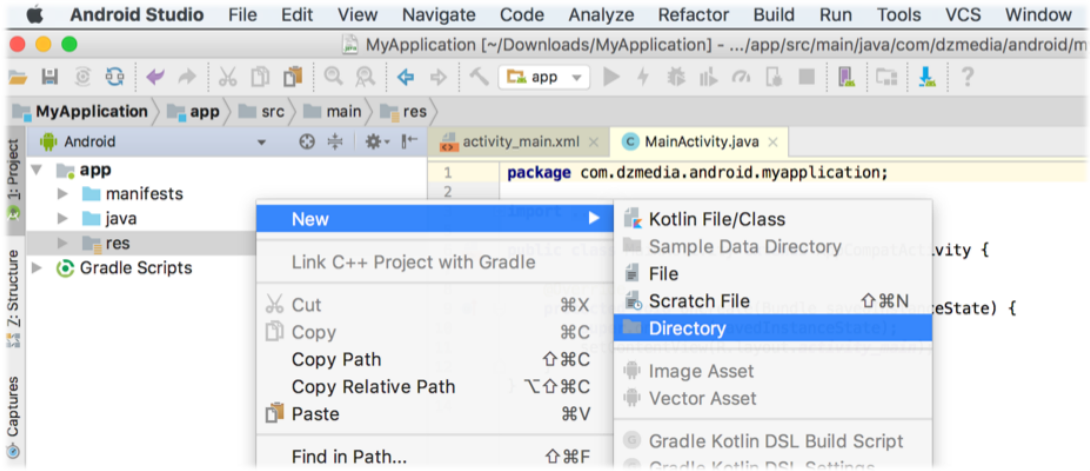
2. Choose the **Download Cloud Config** and then choose **Next**.

   The `awsconfiguration.json` file you download contains the configuration of backend resources that Mobile Hub enabled in your project. Analytics cloud services are enabled for your app by default.



3. Add the backend service configuration file to your app.

   In the Project Navigator, right-click your app's `res` folder, and then choose **New > Directory**. Type `raw` as the directory name and then choose **OK**.



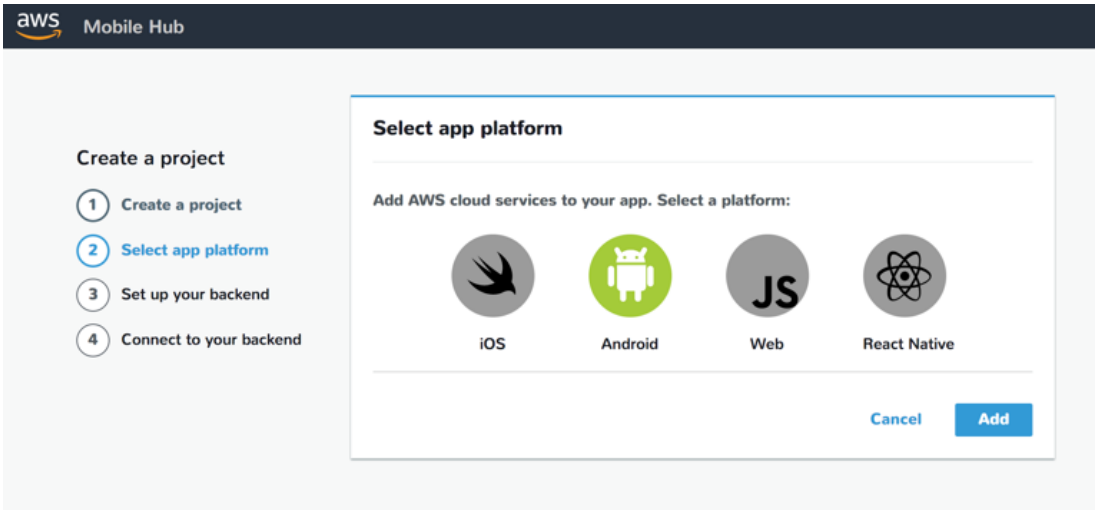
From the location where configuration file, `awsconfiguration.json`, was downloaded in a previous step, drag it into the `res/raw` folder. Android gives a resource ID to any arbitrary file placed in this folder, making it easy to reference in the app.

Remember	Every time you create or update a feature in your Mobile Hub project, download and integrate a new version of your `awsconfiguration.json` into each app in the project that will use the update.

Your backend is now configured. Follow the next steps at Connect to Your Backend (p. 7).

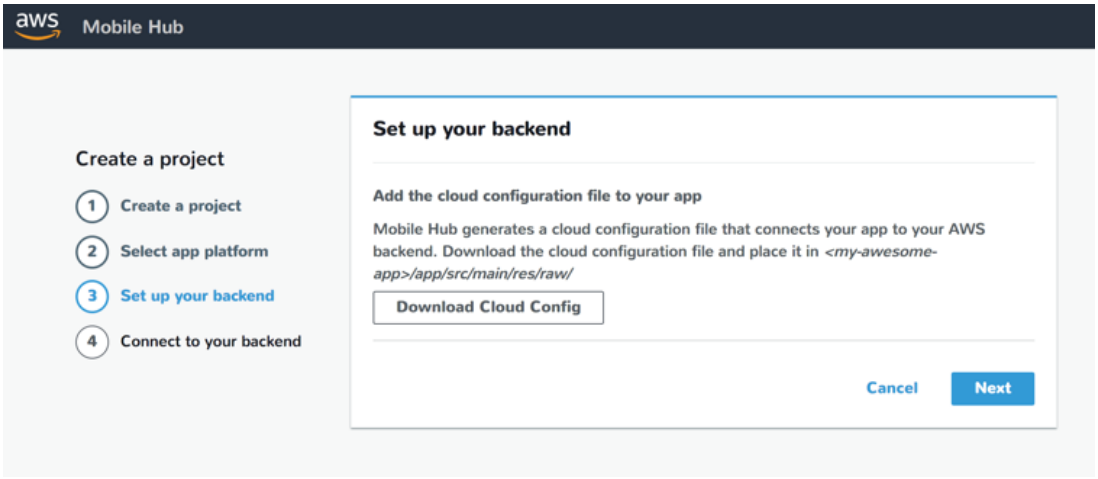
Android - Kotlin

1. Choose **Android** as your platform and then choose Next.



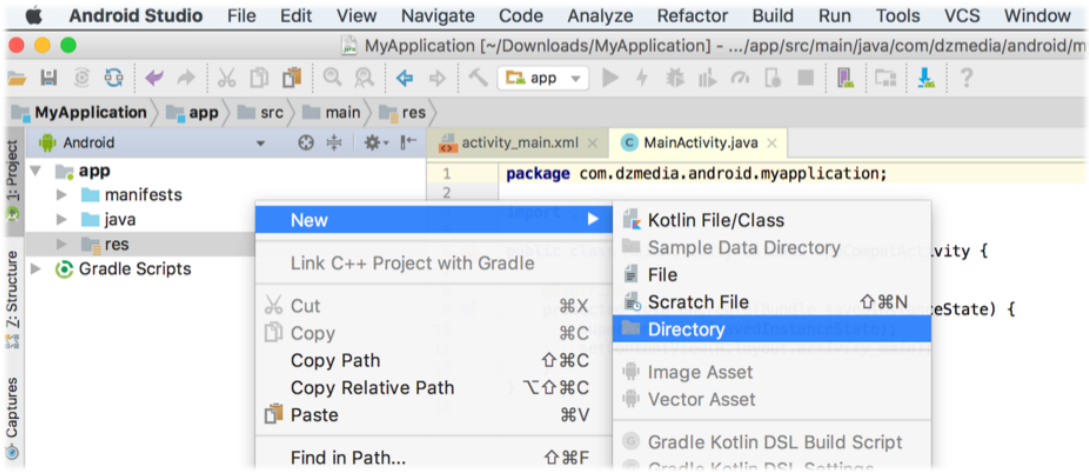
2. Choose the **Download Cloud Config** and then choose **Next**.

   The `awsconfiguration.json` file you download contains the configuration of backend resources that Mobile Hub enabled in your project. Analytics cloud services are enabled for your app by default.



3. Add the backend service configuration file to your app.

   In the Project Navigator, right-click your app's `res` folder, and then choose **New > Directory**. Type `raw` as the directory name and then choose **OK**.



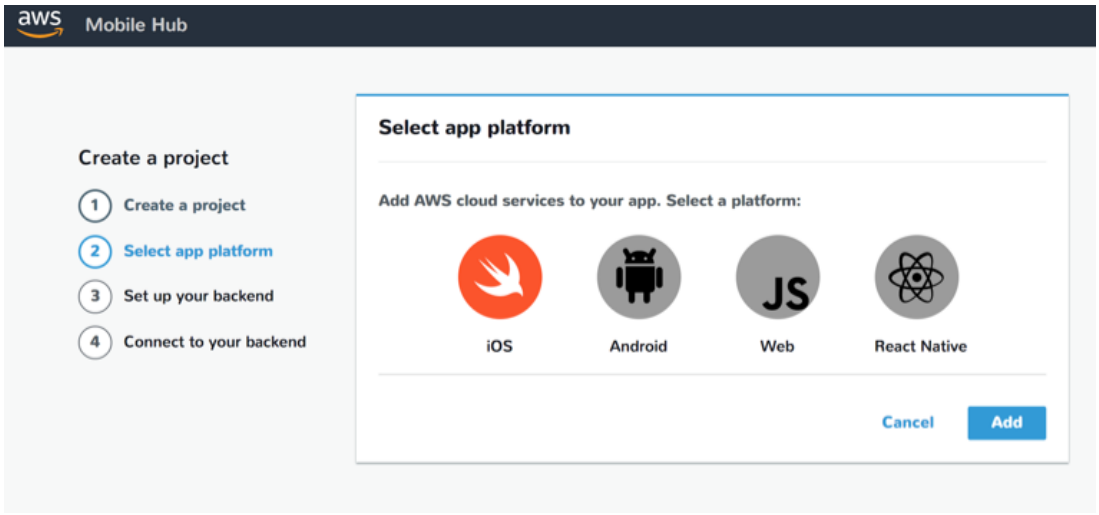
   From the location where configuration file, `awsconfiguration.json`, was downloaded in a previous step, drag it into the `res/raw` folder. Android gives a resource ID to any arbitrary file placed in this folder, making it easy to reference in the app.

Remember	Every time you create or update a feature in your Mobile Hub project, download and integrate a new version of your `awsconfiguration.json` into each app in the project that will use the update.

Your backend is now configured. Follow the next steps at Connect to Your Backend (p. 7).

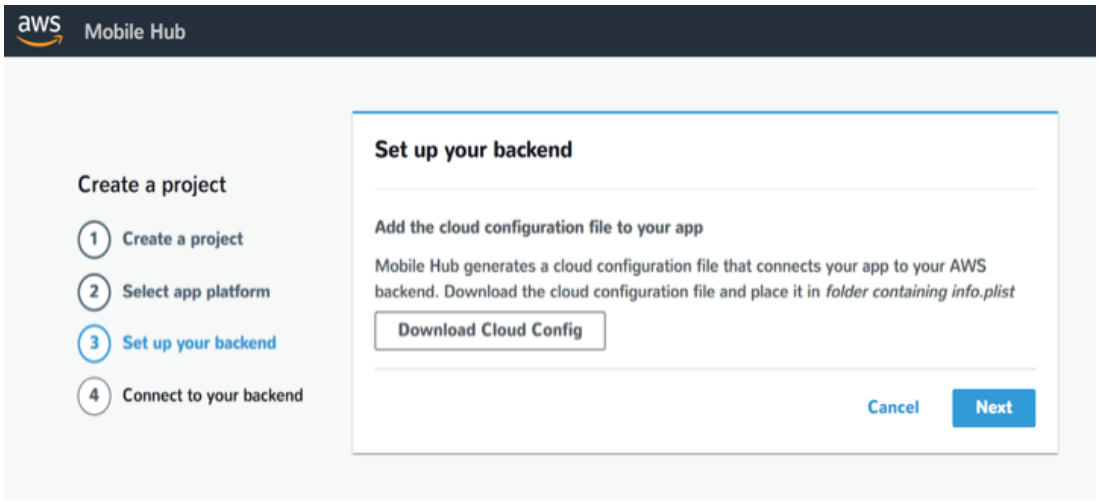
iOS - Swift

1. Pick **iOS** as your platform and choose Next.



2. Choose the **Download Cloud Config** and then choose **Next**.

   The `awsconfiguration.json` file you download contains the configuration of backend resources that Mobile Hub enabled in your project. Analytics cloud services are enabled for your app by default.



3. Add the backend service configuration file to your app.

   From your download location, place `awsconfiguration.json` into the folder containing your `info.plist` file in your Xcode project. Select **Copy items if needed** and **Create groups** in the options dialog. Choose **Next**.

Remember	Every time you create or update a feature in your Mobile Hub project, download and integrate a new version of your `awsconfiguration.json` into each app in the project that will use the update.

Your backend is now configured. Follow the next steps at Connect to Your Backend (p. 7).

Connect to Your Backend

Android - Java

1. Prerequisites
   - Install Android Studio version 2.33 or higher.
   - Install Android SDK v7.11 (Nougat), API level 25.
2. Your `AndroidManifest.xml` must contain:

```
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
```

3. Add dependencies to your `app/build.gradle`, then choose **Sync Now** in the upper right of Android Studio. This libraries enable basic AWS functions, like credentials, and analytics.

```
dependencies {
    implementation ('com.amazonaws:aws-android-sdk-mobile-client:2.6.+@aar')
 { transitive = true }
}
```

4. Add the following code to the `onCreate` method of your main or startup activity. `AWSMobileClient` is a singleton that establishes your connection to <problematic>|AWS|</problematic> and acts as an interface for your services.

```
import com.amazonaws.mobile.client.AWSMobileClient;

  public class YourMainActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        AWSMobileClient.getInstance().initialize(this, new AWSStartupHandler() {
            @Override
            public void onComplete(AWSStartupResult awsStartupResult) {
                Log.d("YourMainActivity", "AWSMobileClient is instantiated and you
 are connected to AWS!");
            }
        }).execute();

        // More onCreate code ...
    }
  }
```

What does this do?	When `AWSMobileClient` is initialized, it constructs the `AWSCredentialsProvider` and `AWSConfiguration` objects which, in turn, are used when creating other SDK clients. The client then makes a Sigv4 signed network call to Amazon Cognito Federated Identities to retrieve AWS credentials that provide the user access to your backend resources. When the network interaction succeeds, the `onComplete` method of the `AWSStartUpHandler` is called.

Your app is now set up to interact with the AWS services you configured in your Mobile Hub project!

Choose the run icon (
<problematic>|play|</problematic>
) in Android Studio to build your app and run it on your device/emulator. Look for `Welcome to AWS!` in your Android Logcat output (choose **View > Tool Windows > Logcat**).

Optional: The following example shows how to retrieve the reference to `AWSCredentialsProvider` and `AWSConfiguration` objects which can be used to instantiate other SDK clients. You can use the `IdentityManager` to fetch the user's AWS identity id either directly from Amazon Cognito or from the locally cached identity id value.

```
import com.amazonaws.auth.AWSCredentialsProvider;
import com.amazonaws.mobile.auth.core.IdentityHandler;
import com.amazonaws.mobile.auth.core.IdentityManager;
import com.amazonaws.mobile.client.AWSMobileClient;
import com.amazonaws.mobile.client.AWSStartupHandler;
import com.amazonaws.mobile.client.AWSStartupResult;
import com.amazonaws.mobile.config.AWSConfiguration;

public class YourMainActivity extends Activity {

    private AWSCredentialsProvider credentialsProvider;
    private AWSConfiguration configuration;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        AWSMobileClient.getInstance().initialize(this, new AWSStartupHandler() {
            @Override
            public void onComplete(AWSStartupResult awsStartupResult) {

                // Obtain the reference to the AWSCredentialsProvider and
 AWSConfiguration objects
                credentialsProvider =
 AWSMobileClient.getInstance().getCredentialsProvider();
                configuration = AWSMobileClient.getInstance().getConfiguration();

                // Use IdentityManager#getUserID to fetch the identity id.
                IdentityManager.getDefaultIdentityManager().getUserID(new
 IdentityHandler() {
                    @Override
                    public void onIdentityId(String identityId) {
                        Log.d("YourMainActivity", "Identity ID = " + identityId);

                        // Use IdentityManager#getCachedUserID to
                        //  fetch the locally cached identity id.
                        final String cachedIdentityId =

 IdentityManager.getDefaultIdentityManager().getCachedUserID();
                    }

                    @Override
                    public void handleError(Exception exception) {
                        Log.d("YourMainActivity", "Error in retrieving the identity"
 + exception);
                    }
                });
            }
        }).execute();

        // .. more code
```

```
    }
}
```

Android - Kotlin

1. Prerequisites
   - Install Android Studio version 2.33 or higher.
   - Install Android SDK v7.11 (Nougat), API level 25.
2. Your `AndroidManifest.xml` must contain:

```
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
```

3. Add dependencies to your `app/build.gradle`, then choose **Sync Now** in the upper right of Android Studio. This libraries enable basic AWS functions, like credentials, and analytics.

```
dependencies {
    implementation ('com.amazonaws:aws-android-sdk-mobile-client:2.6.+@aar')
 { transitive = true }
}
```

4. Add the following code to the `onCreate` method of your main or startup activity. `AWSMobileClient` is a singleton that establishes your connection to <problematic>|AWS|</problematic>
and acts as an interface for your services.

```
import com.amazonaws.mobile.client.AWSMobileClient;

class YourMainActivity : Activity() {
  companion object {
    private val TAG: String = this::class.java.simpleName
  }

  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState);

    AWSMobileClient.getInstance().initialize(this) {
      Log.d(TAG, "AWSMobileClient is initialized")
    }.execute()

    // More onCreate code...
  }
}
```

What does this do?	When `AWSMobileClient` is initialized, it constructs the `AWSCredentialsProvider` and `AWSConfiguration` objects which, in turn, are used when creating other SDK clients. The client then makes a Sigv4 signed network call to Amazon Cognito Federated Identities to retrieve AWS credentials that provide the user access to your backend resources. When the network interaction succeeds, the callback (which is technically the `onComplete` method of the `AWSStartUpHandler`) is called.

Your app is now set up to interact with the AWS services you configured in your Mobile Hub project!

Choose the run icon (
<problematic>|play|</problematic>
) in Android Studio to build your app and run it on your device/emulator. Look for `Welcome to AWS!` in your Android Logcat output (choose **View > Tool Windows > Logcat**).

Optional: The following example shows how to retrieve the reference to `AWSCredentialsProvider` and `AWSConfiguration` objects which can be used to instantiate other SDK clients. You can use the `IdentityManager` to fetch the user's AWS identity id either directly from Amazon Cognito or from the locally cached identity id value.

```
import com.amazonaws.auth.AWSCredentialsProvider
import com.amazonaws.mobile.auth.core.IdentityHandler
import com.amazonaws.mobile.auth.core.IdentityManager
import com.amazonaws.mobile.client.AWSMobileClient
import com.amazonaws.mobile.config.AWSConfiguration

class YourMainActivity : Activity() {
  companion object {
    private val TAG: String = this::class.java.simpleName
  }

  private var credentialsProvider: AWSCredentialsProvider? = null
  private var awsConfiguration: AWSConfiguration? = null

  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState);

    AWSMobileClient.getInstance().initialize(this) {
      credentialsProvider = AWSMobileClient.getInstance().credentialsProvider
      awsConfiguration = AWSMobileClient.getInstance().configuration

      IdentityManager.getDefaultIdentityManager().getUserID(object : IdentityHandler
 {
        override fun handleError(exception: Exception?) {
          Log.e(TAG, "Retrieving identity: ${exception.message}")
        }

        override fun onIdentityId(identityId: String?) {
          Log.d(TAG, "Identity = $identityId")
          val cachedIdentityId =
 IdentityManager.getDefaultIdentityManager().cachedUserID
          // Do something with the identity here
        }
      })
    }.execute()

    // More onCreate code...
  }
}
```

iOS - Swift

1. Prerequisites
   - Install Xcode version 8.0 or later.
2. Install Cocoapods. From a terminal window run:

```
sudo gem install cocoapods
```

3. Create `Podfile`. From a terminal window, navigate to the directory that contains your project's `.xcodeproj` file and run:

```
pod init
```

4. Add core AWS Mobile SDK components to your build.

```
platform :ios, '9.0'
target :'YOUR-APP-NAME' do
    use_frameworks!
    pod 'AWSMobileClient', '~> 2.6.13'
    # other pods
end
```

5. Install dependencies by runnng:

```
pod install --repo-update
```

   If you encounter an error message that begins "`[!] Failed to connect to GitHub to update the CocoaPods/Specs . . .`", and your internet connectivity is working, you may need to update openssl and Ruby.

6. The command `pod install` creates a new workspace file. Close your Xcode project and reopen it using `./YOUR-PROJECT-NAME.xcworkspace`.

Use **ONLY** your .xcworkspace	Remember to always use `./YOUR-PROJECT-NAME.xcworkspace` to open your Xcode project from now on.

7. Rebuild your app after reopening it in the workspace to resolve APIs from new libraries called in your code. This is a good practice any time you add import statements.
8. Replace the `return true` statement in `didFinishLaunching` with the following code in your AppDelegate to establish a run-time connection with AWS Mobile.

```
import UIKit
import AWSMobileClient

@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate {

func application(_ application: UIApplication,
                 didFinishLaunchingWithOptions launchOptions:
 [UIApplicationLaunchOptionsKey: Any]?) -> Bool {
    // Override point for customization after application launch.

    // Create AWSMobileClient to connect with AWS
    return AWSMobileClient.sharedInstance().interceptApplication(
        application,
        didFinishLaunchingWithOptions: launchOptions)

}
```

What does this do?	When `AWSMobileClient` is initialized, it makes a Sigv4 signed network call to Amazon Cognito Federated Identities to retrieve AWS credentials that provide the user access to your backend resources. When the network

	interaction succeeds, the `onComplete` method of the `AWSStartUpHandler` is called.

Your app is now set up to interact with the AWS services you configured in your Mobile Hub project!

Choose the run icon (
<problematic>|play|</problematic>
) in the top left of the Xcode window or type
<problematic>|Acommand|</problematic>
-R to build and run your app. Look for `Welcome to AWS!` in the log output.

Optional: If you want to make sure you're connected to AWS, import `AWSCore` and add the following code to `didFinishLaunchingWithOptions` before you return `AWSMobileClient`.

```
import AWSCore

      //. . .

AWSDDLog.add(AWSDDTTYLogger.sharedInstance)
AWSDDLog.sharedInstance.logLevel = .info
```

Optional: The following example shows how to retrieve the reference to `AWSCredentialsProvider` object which can be used to instantiate other SDK clients. You can use the `AWSIdentityManager` to fetch the AWS identity id of the user from Amazon Cognito.

```
import UIKit
import AWSMobileClient
import AWSAuthCore

class ViewController: UIViewController {

    @IBOutlet weak var textfield: UITextField!
    override func viewDidLoad() {
        super.viewDidLoad()
        textfield.text = "View Controller Loaded"

        // Get the AWSCredentialsProvider from the AWSMobileClient
        let credentialsProvider =
 AWSMobileClient.sharedInstance().getCredentialsProvider()

        // Get the identity Id from the AWSIdentityManager
        let identityId = AWSIdentityManager.default().identityId
    }
}
```

Next Steps

- Add Analytics (p. 13)
- Add User Sign-in (p. 20)
- Add Push Notification (p. 43)
- Add NoSQL Database (p. 54)
- Add User File Storage (p. 66)
- Add Cloud logic (p. 75)

- Add Messaging (p. 83)
- Add Conversational Bots (p. 84)
- Add Hosting and Streaming

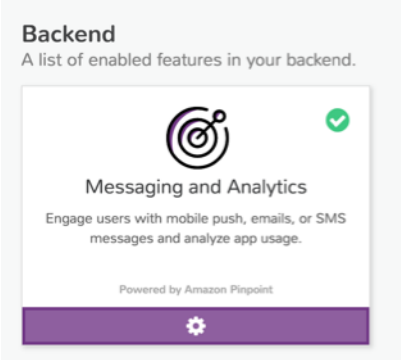
Add Analytics to your Mobile App with Amazon Pinpoint

Overview

Gather the data that helps improve your app's usability, monetization, and engagement with your users. Mobile Hub deploys your analytics backend when you enable the Messaging and Analytics (p. 340) feature, which uses the Amazon Pinpoint service.

Set up your Backend

1. Complete the Get Started (p. 2) steps before your proceed.
2. When you create a project, we enable analytics by default in your backend. You should see a green check mark present on the **Analytics** tile in your backend, indicating that the feature is enabled. If the check mark is absent, choose **Analytics**, and then choose **Enable**.



Connect to your Backend

Use the following steps to add analytics to your mobile app and monitor the results through Amazon Pinpoint.

Add Analytics

Android - Java

1. Set up AWS Mobile SDK components by following the Basic Backend Setup (p. 2) steps. These include:
   a. Include the following libraries in your `app/build.gradle` dependencies list.

```
dependencies{
   implementation 'com.amazonaws:aws-android-sdk-pinpoint:2.6.+'
   implementation ('com.amazonaws:aws-android-sdk-mobile-client:2.6.+@aar')
 { transitive = true }
   // other dependencies . . .
```

```
}
```

- `aws-android-sdk-pinpoint` library enables sending analytics to Amazon Pinpoint.
- `aws-android-sdk-mobile-client` library gives access to the AWS credentials provider and configurations.

b. Add required permissions to your app manifest.

The AWS Mobile SDK required the `INTERNET` and `ACCESS_NETWORK_STATE` permissions. These are defined in the `AndroidManifest.xml` file.

```
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
```

2. Add calls to capture session starts and stops.

Three typical places to instrument your app session start and stop are:

- Start a session in the `Application.onCreate()` method.
- Start a session in the `onCreate()` method of the app's first activity.
- Start and/or stop a session in the ActivityLifecycleCallbacks class.

The following example shows starting a session in the `OnCreate` event of `MainActivity`.

```
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

import com.amazonaws.mobileconnectors.pinpoint.PinpointManager;
import com.amazonaws.mobileconnectors.pinpoint.PinpointConfiguration;
import com.amazonaws.mobile.client.AWSMobileClient;

public class MainActivity extends AppCompatActivity {

    public static PinpointManager pinpointManager;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        // Initialize the AWS Mobile Client
        AWSMobileClient.getInstance().initialize(this).execute();

        PinpointConfiguration config = new PinpointConfiguration(
                MainActivity.this,
                AWSMobileClient.getInstance().getCredentialsProvider(),
                AWSMobileClient.getInstance().getConfiguration()
        );
        pinpointManager = new PinpointManager(config);
        pinpointManager.getSessionClient().startSession();
        pinpointManager.getAnalyticsClient().submitEvents();
    }
}
```

To stop the session, use `stopSession()` and `submitEvents()` at the last point in the session you want to capture.

```
// . . .

pinpointManager.getSessionClient().stopSession();
pinpointManager.getAnalyticsClient().submitEvents();
```

```
// . . .
```

Android - Kotlin

1. Set up AWS Mobile SDK components by following the Basic Backend Setup (p. 2) steps. These include:

   a. Include the following libraries in your `app/build.gradle` dependencies list.

   ```
   dependencies {
      implementation 'com.amazonaws:aws-android-sdk-pinpoint:2.6.+'
      implementation ('com.amazonaws:aws-android-sdk-mobile-client:2.6.+@aar')
    { transitive = true }
      // other dependencies . . .
   }
   ```

   - `aws-android-sdk-pinpoint` library enables sending analytics to Amazon Pinpoint.
   - `aws-android-sdk-mobile-client` library gives access to the AWS credentials provider and configurations.

   b. Add required permissions to your app manifest.

   The AWS Mobile SDK required the `INTERNET` and `ACCESS_NETWORK_STATE` permissions. These are defined in the `AndroidManifest.xml` file.

   ```
   <uses-permission android:name="android.permission.INTERNET"/>
   <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
   ```

2. Add calls to capture session starts and stops.

   Three typical places to instrument your app session start and stop are:

   - Start a session in the `Application.onCreate()` method.
   - Start a session in the `onCreate()` method of the app's first activity.
   - Start and/or stop a session in the ActivityLifecycleCallbacks class.

   The following example shows starting a session in the `OnCreate` event of `MainActivity`.

   ```
   import android.support.v7.app.AppCompatActivity;
   import android.os.Bundle;
   import com.amazonaws.mobileconnectors.pinpoint.PinpointManager;
   import com.amazonaws.mobileconnectors.pinpoint.PinpointConfiguration;
   import com.amazonaws.mobile.client.AWSMobileClient;

   class MainActivity : AppCompatActivity() {
       companion object {
           var pinpointManager: PinpointManager? = null
       }

       override fun onCreate(savedInstanceState: Bundle?) {
           super.onCreate(savedInstanceState)
           setContentView(R.layout.activity_main)

           // Initialize the AWS Mobile client
           AWSMobileClient.getInstance().initialize(this).execute()

           with (AWSMobileClient.getInstance()) {
               val config = PinpointConfiguration(this, credentialsProvider,
    configuration)
               pinpointManager = PinpointManager(config)
   ```

```
        }

        pinpointManager?.sessionClient?.startSession()
        pinpointManager?.analyticsClient?.submitEvents()
    }
}
```

To stop the session, use `stopSession()` and `submitEvents()` at the last point in the session you want to capture.

```
// . . .

pinpointManager?.sessionClient?.stopSession();
pinpointManager?.analyticsClient?.submitEvents();

// . . .
```

iOS - Swift

1. Set up AWS Mobile SDK components with the following Basic Backend Setup (p. 2) steps.
   a. The `Podfile` that you configure to install the AWS Mobile SDK must contain:

```
platform :ios, '9.0'
target :'YourAppName' do
  use_frameworks!

    pod 'AWSPinpoint', '~> 2.6.13'

    # other pods

end
```

   Run `pod install --repo-update` before you continue.

   If you encounter an error message that begins "`[!] Failed to connect to GitHub to update the CocoaPods/Specs . . .`", and your internet connectivity is working, you may need to update openssl and Ruby.

   b. Classes that call Amazon Pinpoint APIs must use the following import statements:

```
import AWSCore
import AWSPinpoint
```

   c. Insert the following code into the `application(_:didFinishLaunchingWithOptions:)` method of your app's `AppDelegate.swift`.

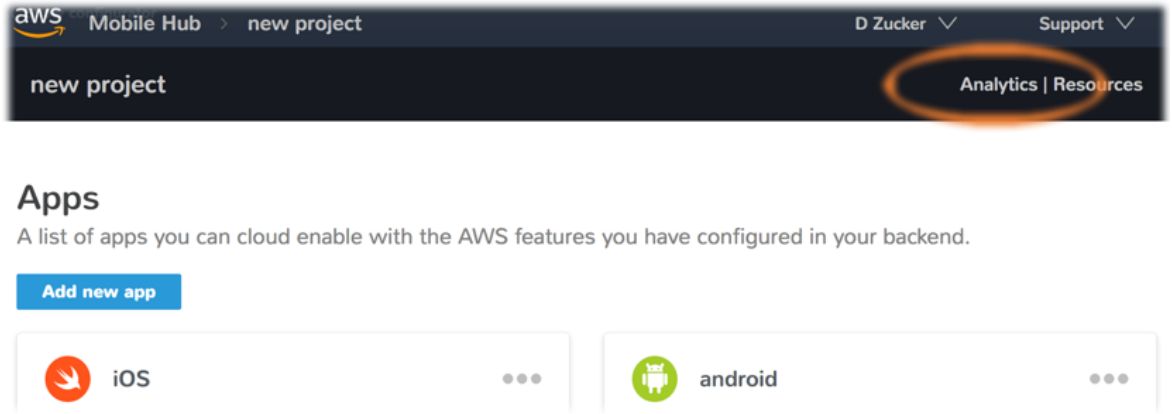
```
class AppDelegate: UIResponder, UIApplicationDelegate {

    var pinpoint: AWSPinpoint?

    func application(_ application: UIApplication, didFinishLaunchingWithOptions
 launchOptions:
    [UIApplicationLaunchOptionsKey: Any]?) -> Bool {

    //. . .

    // Initialize Pinpoint
    pinpoint = AWSPinpoint(configuration:
```

```
            AWSPinpointConfiguration.defaultPinpointConfiguration(launchOptions:
 launchOptions))

    //. . .
    }
}
```

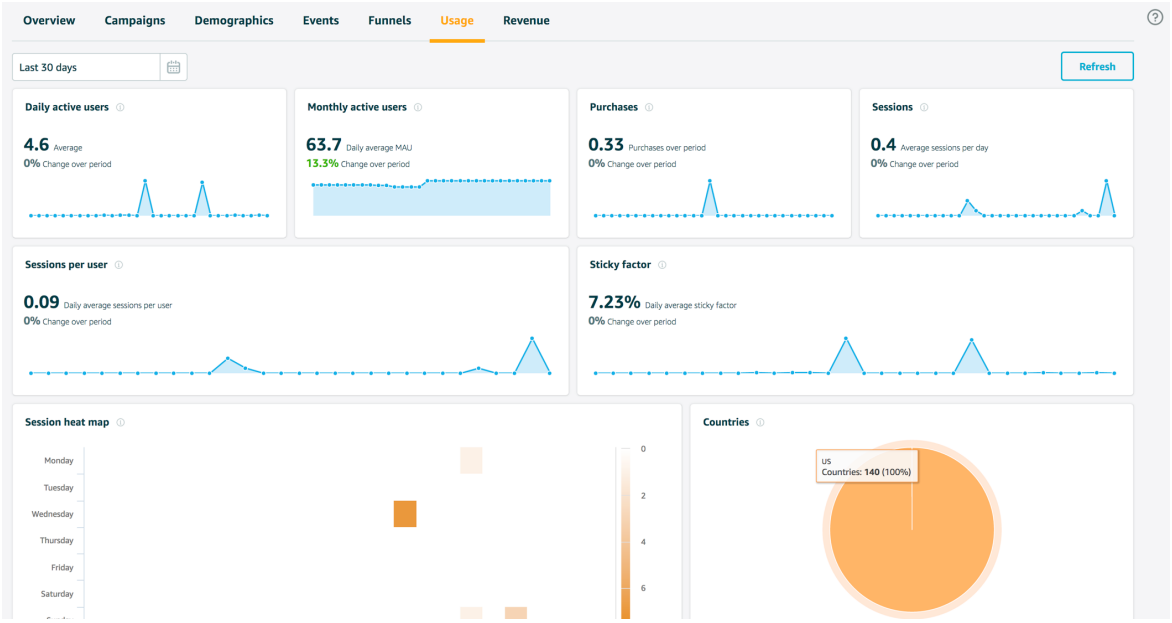
Monitor Analytics

Build and run your app to see usage metrics in Amazon Pinpoint.

1. To see visualizations of the analytics coming from your app, open your project in the Mobile Hub console.
2. Choose **Analytics** on the upper right to open the Amazon Pinpoint console.



1. Choose **Analytics** from the icons on the left of the console, and view the graphs of your app's usage. It may take up to 15 minutes for metrics to become visible.



Learn more about Amazon Pinpoint.

Enable Custom App Analytics

Instrument your code to capture app usage event information, including attributes you define. Use graphs of your custom usage event data in the Amazon Pinpoint console. Visualize how your users' behavior aligns with a model you design using Amazon Pinpoint Funnel Analytics, or use stream the data for deeper analysis.

Use the following steps to implement Amazon Pinpoint custom analytics for your app.

Android - Java

```
import com.amazonaws.mobileconnectors.pinpoint.analytics.AnalyticsEvent;

public void logEvent() {
    final AnalyticsEvent event =
        pinpointManager.getAnalyticsClient().createEvent("EventName")
            .withAttribute("DemoAttribute1", "DemoAttributeValue1")
            .withAttribute("DemoAttribute2", "DemoAttributeValue2")
            .withMetric("DemoMetric1", Math.random());

    pinpointManager.getAnalyticsClient().recordEvent(event);
    pinpointManager.getAnalyticsClient().submitEvents();
}
```

Android - Kotlin

```
import com.amazonaws.mobileconnectors.pinpoint.analytics.AnalyticsEvent;

fun logEvent() {
    pintpointManager?.analyticsClient?.let {
        val event = it.createEvent("EventName")
            .withAttribute("DemoAttribute1", "DemoAttributeValue1")
            .withAttribute("DemoAttribute2", "DemoAttributeValue2")
            .withMetric("DemoMetric1", Math.random());
        it.recordEvent(event)
        it.submitEvents()
}
```

iOS - Swift

```
func logEvent() {

    let pinpointAnalyticsClient =
        AWSPinpoint(configuration:
            AWSPinpointConfiguration.defaultPinpointConfiguration(launchOptions:
 nil)).analyticsClient

    let event = pinpointAnalyticsClient.createEvent(withEventType: "EventName")
    event.addAttribute("DemoAttributeValue1", forKey: "DemoAttribute1")
    event.addAttribute("DemoAttributeValue2", forKey: "DemoAttribute2")
    event.addMetric(NSNumber.init(value: arc4random() % 65535), forKey: "EventName")
    pinpointAnalyticsClient.record(event)
    pinpointAnalyticsClient.submitEvents()

}
```

Build, run, and try your app, and then view your custom events in the **Events** tab of the Amazon Pinpoint console (use your Mobile Hub project / **Analytics** > Amazon Pinpoint console / **Analytics** > **Events**). Look for the name of your event in the **Events** dropdown menu.

Enable Revenue Analytics

Amazon Pinpoint supports the collection of monetization event data. Use the following steps to place and design analytics related to purchases through your app.

Android - Java

```
import
 com.amazonaws.mobileconnectors.pinpoint.analytics.monetization.AmazonMonetizationEventBuilder;

public void logMonetizationEvent() {
    final AnalyticsEvent event =
        AmazonMonetizationEventBuilder.create(pinpointManager.getAnalyticsClient())
            .withFormattedItemPrice("$10.00")
            .withProductId("DEMO_PRODUCT_ID")
            .withQuantity(1.0)
            .withProductId("DEMO_TRANSACTION_ID").build();

    pinpointManager.getAnalyticsClient().recordEvent(event);
    pinpointManager.getAnalyticsClient().submitEvents();
}
```

Android - Kotlin

```
import
 com.amazonaws.mobileconnectors.pinpoint.analytics.monetization.AmazonMonetizationEventBuilder;

public void logMonetizationEvent() {
    pinpointManager?.analyticsClient?.let {
        val event = AmazonMonetizationEventBuilder.create(it)
           .withFormattedItemPrice("$10.00")
           .withProductId("DEMO_PRODUCT_ID")
           .withQuantity(1.0)
           .withProductId("DEMO_TRANSACTION_ID").build();
        it.recordEvent(event)
        it.submitEvents()
    }
}
```

iOS - Swift

```
func sendMonetizationEvent()
 {
     let pinpointClient = AWSPinpoint(configuration:
         AWSPinpointConfiguration.defaultPinpointConfiguration(launchOptions: nil))

     let pinpointAnalyticsClient = pinpointClient.analyticsClient

     let event =
         pinpointAnalyticsClient.createVirtualMonetizationEvent(withProductId:
             "DEMO_PRODUCT_ID", withItemPrice: 1.00, withQuantity: 1, withCurrency:
 "USD")
     pinpointAnalyticsClient.record(event)
     pinpointAnalyticsClient.submitEvents()
 }
```

Add User Sign-in to Your Mobile App with Amazon Cognito

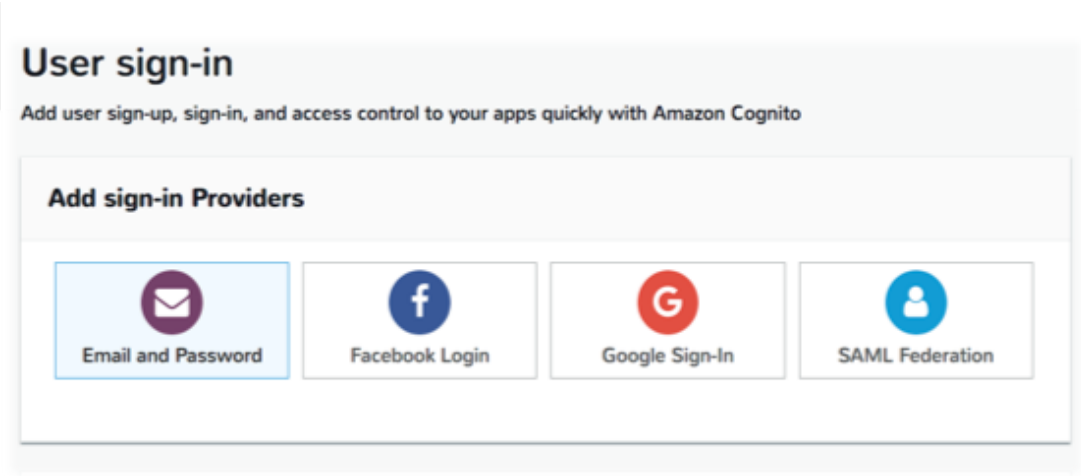
Enable your users to sign-in using credentials from Facebook, Google, or your own custom user directory. The AWS Mobile Hub User Sign-in (p. 348) feature is powered by Amazon Cognito, and the SDK provides a pre-built, configurable (p. 179) Sign-in UI based on the identity provider(s) you configure.

Set Up Your Backend

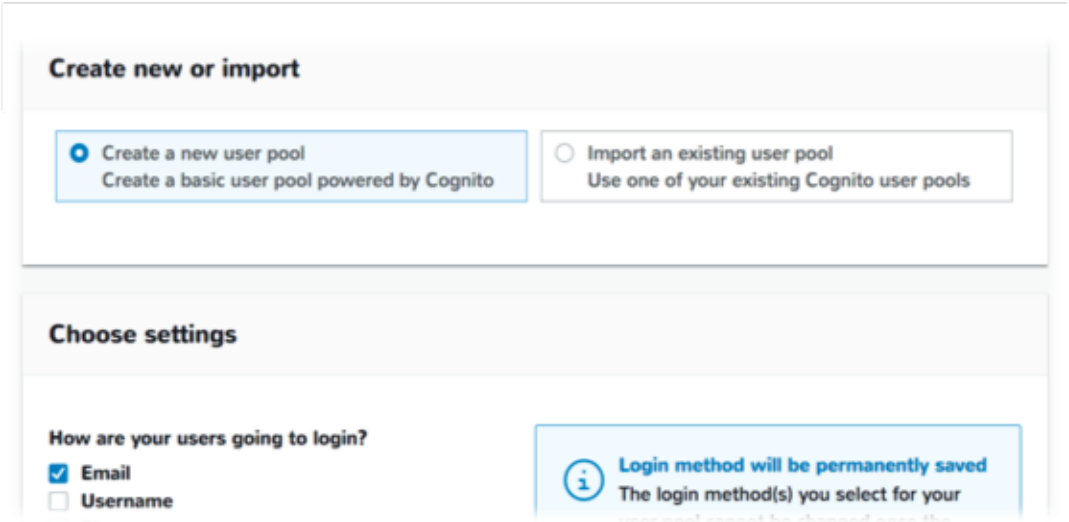
Prerequisite Complete the Get Started (p. 2) steps before your proceed.

Email & Password

1. Enable **User Sign-in**: Open your project in Mobile Hub console and choose the **User Sign-in** tile.
2. Choose **Email and Password sign-in**

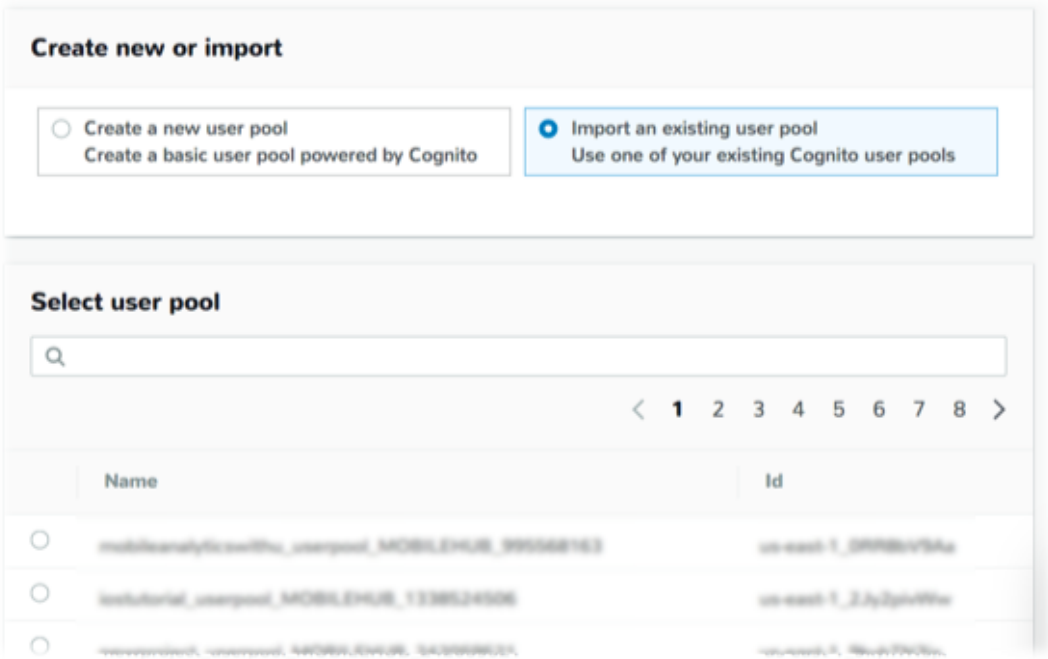


- Choose **Create a new user pool**, the feature and then select sign-in settings including: allowed login methods; multi-factor authentication; and password requirements. Then choose **Create user pool**.

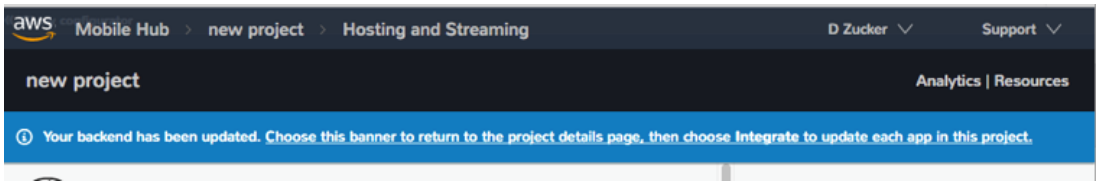


Or:

- Choose **Import an existing user pool**, select a user pool from the list of pools that are available in the account. Choose if sign-in is required, and then choose **Create user pool**. If you import a user pool that is in use by another app, then the two apps will share the user directory and authenticate sign-in by the same set of users.

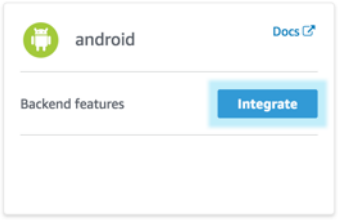


3. When you are done configuring providers, choose **Click here to return to project details page** in the blue banner at the top.



4. On the project detail page, choose the flashing **Integrate** button, and then complete the steps that guide you to connect your backend.

   If your project contains apps for more than one platform, any that need to complete those steps will also display a flashing **Integrate** button. The reminder banner will remain in place until you have taken steps to update the configuration of each app in the project.

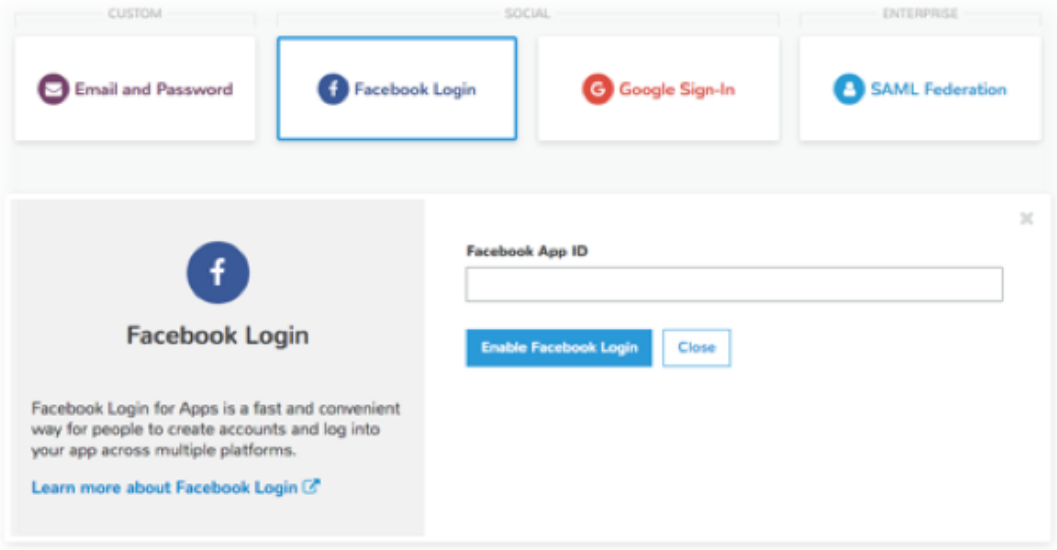


5. Follow the Set up Email & Password Login (p. 23) steps to connect to your backend from your app.

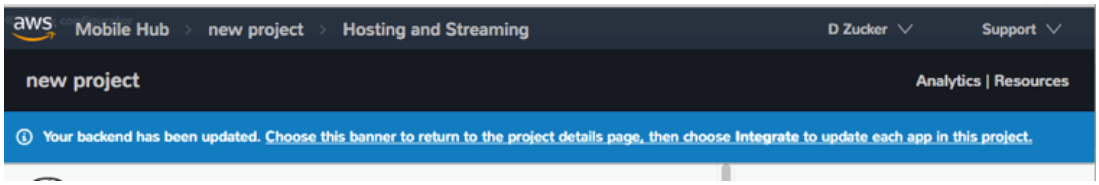
Facebook

1. Enable **User Sign-in**: Open your project in Mobile Hub console and choose the **User Sign-in** tile.

2. **Configure Facebook sign-in**: Choose the feature and then type your Facebook App ID and then choose **Enable Facebook login**. To retrieve or create your Facebook App ID, see Setting Up Facebook Authentication..

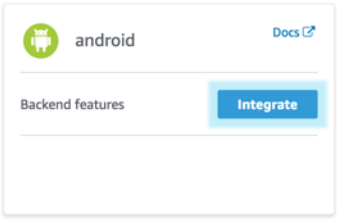


3. When you are done configuring providers, choose **Click here to return to project details page** in the blue banner at the top.



4. On the project detail page, choose the flashing **Integrate** button, and then complete the steps that guide you to connect your backend.

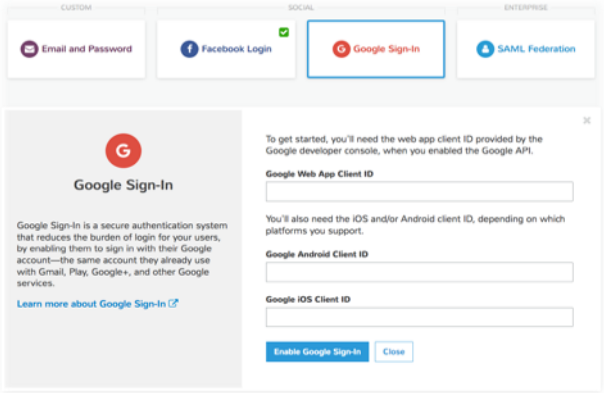
   If your project contains apps for more than one platform, any that need to complete those steps will also display a flashing **Integrate** button. The reminder banner will remain in place until you have taken steps to update the configuration of each app in the project.



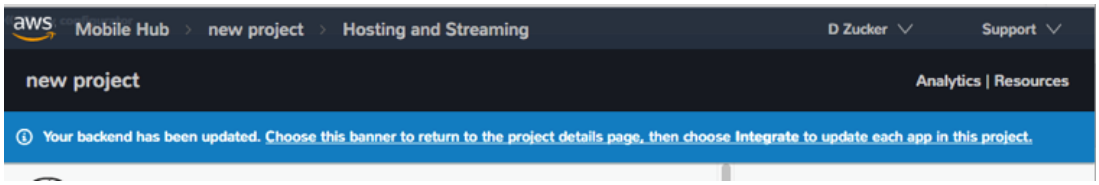
5. Follow the steps at Set Up Facebook Login (p. 29) to connect to your backend from your app.

Google

1. Enable **User Sign-in**: Open your project in Mobile Hub console and choose the **User Sign-in** tile.
2. Configure **Google sign-in**: Choose the feature and then type in your Google Web App Client ID, and the Google Android or iOS Client ID (or both), and then choose Enable Google Sign-In. To retrieve or create your Google Client IDs, see Setting Up Google Authentication.

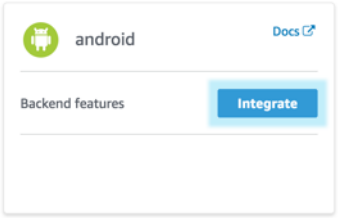


3. When you are done configuring providers, choose **Click here to return to project details page** in the blue banner at the top.



4. On the project detail page, choose the flashing **Integrate** button, and then complete the steps that guide you to connect your backend.

   If your project contains apps for more than one platform, any that need to complete those steps will also display a flashing **Integrate** button. The reminder banner will remain in place until you have taken steps to update the configuration of each app in the project.



5. Follow the steps at Set Up Google Login (p. 36) to connect to your backend from your app.

Setup Email & Password Login in your Mobile App

Choose your platform:

Android - Java

Use Android API level 23 or higher	The AWS Mobile SDK library for Android sign-in (`aws-android-sdk-auth-ui`) provides the activity and view for presenting a `SignInUI` for the sign-in providers you configure. This library depends on the Android SDK API Level 23 or higher.

1. Add or update your AWS backend configuration file to incorporate your new sign-in. For details, see the last steps in the Get Started: Set Up Your Backend (p. 2) section.

2. Add these permisions to the `AndroidManifest.xml` file:

```
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
```

3. Add these dependencies to the `app/build.gradle` file:

```
dependencies {
     // Mobile Client for initializing the SDK
     implementation ('com.amazonaws:aws-android-sdk-mobile-client:2.6.+@aar')
 { transitive = true }

     // Cognito UserPools for SignIn
     implementation 'com.android.support:support-v4:24.+'
     implementation ('com.amazonaws:aws-android-sdk-auth-userpools:2.6.+@aar')
 { transitive = true }

     // Sign in UI Library
     implementation 'com.android.support:appcompat-v7:24.+'
     implementation ('com.amazonaws:aws-android-sdk-auth-ui:2.6.+@aar') { transitive
 = true }
}
```

4. Create an activity that will present your sign-in screen.

   In Android Studio, choose **File > New > Activity > Basic Activity** and type an activity name, such as `AuthenticatorActivity`. If you want to make this your starting activity, move the the intent filter block containing `.LAUNCHER` to the `AuthenticatorActivity` in your app's `AndroidManifest.xml`.

```
<activity android:name=".AuthenticatorActivity">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

5. Update the `onCreate` function of your `AuthenticatorActivity` to call `AWSMobileClient`. This component provides the functionality to resume a signed-in authentication session. It makes a network call to retrieve the AWS credentials that allow users to access your AWS resources and registers a callback for when that transaction completes.

   If the user is signed in, the app goes to the `NextActivity`, otherwise it presents the user with the AWS Mobile ready made, configurable sign-in UI. `NextActivity` `Activity` class a user sees after a successful sign-in.

```
import android.app.Activity;
import android.os.Bundle;

import com.amazonaws.mobile.auth.ui.SignInUI;
import com.amazonaws.mobile.client.AWSMobileClient;
import com.amazonaws.mobile.client.AWSStartupHandler;
import com.amazonaws.mobile.client.AWSStartupResult;

public class AuthenticatorActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_authenticator);

        // Add a call to initialize AWSMobileClient
        AWSMobileClient.getInstance().initialize(this, new AWSStartupHandler() {
```

```
            @Override
            public void onComplete(AWSStartupResult awsStartupResult) {
                SignInUI signin = (SignInUI)
 AWSMobileClient.getInstance().getClient(AuthenticatorActivity.this, SignInUI.class);
                signin.login(AuthenticatorActivity.this,
 NextActivity.class).execute();
            }
        }).execute();
    }
}
```

Choose the run icon (
<problematic>|play|</problematic>
) in Android Studio to build your app and run it on your device/emulator. You should see our ready made sign-in UI for your app. Checkout the next steps to learn how to customize your UI (p. 179).

API References	• AWSMobileClient A library that initializes the SDK, constructs CredentialsProvider and AWSConfiguration objects, fetches the AWS credentials, and creates a SDK SignInUI client instance. • Auth UserPools A wrapper Library for Amazon Cognito UserPools that provides a managed Email/Password sign-in UI. • Auth Core A library that caches and federates a login provider authentication token using Amazon Cognito Federated Identities, caches the federated AWS credentials, and handles the sign-in flow.

Android - Kotlin

Use Android API level 23 or higher	The AWS Mobile SDK library for Android sign-in (`aws-android-sdk-auth-ui`) provides the activity and view for presenting a `SignInUI` for the sign-in providers you configure. This library depends on the Android SDK API Level 23 or higher.

1. Add or update your AWS backend configuration file to incorporate your new sign-in. For details, see the last steps in the Get Started: Set Up Your Backend (p. 2) section.
2. Add these permisions to the `AndroidManifest.xml` file:

```
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
```

3. Add these dependencies to the `app/build.gradle` file:

```
dependencies {
     // Mobile Client for initializing the SDK
```

```
    implementation ('com.amazonaws:aws-android-sdk-mobile-client:2.6.+@aar')
{ transitive = true }

    // Cognito UserPools for SignIn
    implementation 'com.android.support:support-v4:24.+'
    implementation ('com.amazonaws:aws-android-sdk-auth-userpools:2.6.+@aar')
{ transitive = true }

    // Sign in UI Library
    implementation 'com.android.support:appcompat-v7:24.+'
    implementation ('com.amazonaws:aws-android-sdk-auth-ui:2.6.+@aar') { transitive
= true }
}
```

4. Create an activity that will present your sign-in screen.

   In Android Studio, choose **File > New > Activity > Basic Activity** and type an activity name, such as `AuthenticatorActivity`. If you want to make this your starting activity, move the the intent filter block containing `.LAUNCHER` to the `AuthenticatorActivity` in your app's `AndroidManifest.xml`.

```
<activity android:name=".AuthenticatorActivity">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

5. Update the `onCreate` function of your `AuthenticatorActivity` to call `AWSMobileClient`. This component provides the functionality to resume a signed-in authentication session. It makes a network call to retrieve the AWS credentials that allow users to access your AWS resources and registers a callback for when that transaction completes.

   If the user is signed in, the app goes to the `NextActivity`, otherwise it presents the user with the AWS Mobile ready made, configurable sign-in UI. `NextActivity` `Activity` class a user sees after a successful sign-in.

```
import android.app.Activity;
import android.os.Bundle;

import com.amazonaws.mobile.auth.ui.SignInUI;
import com.amazonaws.mobile.client.AWSMobileClient;
import com.amazonaws.mobile.client.AWSStartupHandler;
import com.amazonaws.mobile.client.AWSStartupResult;

class AuthenticatorActivity : Activity() {
  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)

    AWSMobileClient.getInstance().initialize(this) {
      val ui = AWSMobileClient.getInstance().getClient(this@AuthenticatorActivity,
 SignInUI::class.java)
      ui.login(this@AuthenticatorActivity, NextActivity::class.java).execute()
    }.execute()
  }
}
```

Choose the run icon (
<problematic>|play|</problematic>
) in Android Studio to build your app and run it on your device/emulator. You should see our ready made sign-in UI for your app. Checkout the next steps to learn how to customize your UI (p. 179).

API References	• AWSMobileClient A library that initializes the SDK, constructs CredentialsProvider and AWSConfiguration objects, fetches the AWS credentials, and creates a SDK SignInUI client instance. • Auth UserPools A wrapper Library for Amazon Cognito UserPools that provides a managed Email/Password sign-in UI. • Auth Core A library that caches and federates a login provider authentication token using Amazon Cognito Federated Identities, caches the federated AWS credentials, and handles the sign-in flow.

iOS - Swift

1. Add or update your AWS backend configuration file to incorporate your new sign-in. For details, see the last steps in the Get Started: Set Up Your Backend (p. 2) section.
2. Add the following dependencies in your project's `Podfile`.

```
platform :ios, '9.0'
target :'YOUR-APP-NAME' do
    use_frameworks!
    pod 'AWSUserPoolsSignIn', '~> 2.6.13'
    pod 'AWSAuthUI', '~> 2.6.13'
    pod 'AWSMobileClient', '~> 2.6.13'
    # other pods
end
```

3. Pull the SDK libraries into your local repo.

   If you encounter an error message that begins "`[!] Failed to connect to GitHub to update the CocoaPods/Specs . . .`", and your internet connectivity is working, you may need to update openssl and Ruby.
4. Create a AWSMobileClient and initialize the SDK.

   Add code to create an instance of `AWSMobileClient` in the `application:open url` function of your `AppDelegate.swift`, to resume a previously signed-in authenticated session.

   Then add another instance of `AWSMobileClient` in the `didFinishLaunching` function to register the sign in providers, and to fetch an Amazon Cognito credentials that AWS will use to authorize access once the user signs in.

```
import UIKit

import AWSMobileClient

@UIApplicationMain

class AppDelegate: UIResponder, UIApplicationDelegate {

    // Add a AWSMobileClient call in application:open url
    func application(_ application: UIApplication, open url: URL,
        sourceApplication: String?, annotation: Any) -> Bool {
```

```
        return AWSMobileClient.sharedInstance().interceptApplication(
            application, open: url,
            sourceApplication: sourceApplication,
            annotation: annotation)

    }

    // Add a AWSMobileClient call in application:didFinishLaunching
     func application(
        _ application: UIApplication,
            didFinishLaunchingWithOptions launchOptions:
                [UIApplicationLaunchOptionsKey: Any]?) -> Bool {

         return AWSMobileClient.sharedInstance().interceptApplication(
             application, didFinishLaunchingWithOptions:
             launchOptions)
    }

    // Other functions in AppDelegate . . .

  }
```

5. Implement your sign-in UI by calling the library provided in the SDK.

```
import UIKit
import AWSAuthCore
import AWSAuthUI

class SampleViewController: UIViewController {

    override func viewDidLoad() {

        super.viewDidLoad()

        if !AWSSignInManager.sharedInstance().isLoggedIn {
           AWSAuthUIViewController
             .presentViewController(with: self.navigationController!,
                  configuration: nil,
                  completionHandler: { (provider: AWSSignInProvider, error: Error?)
 in
                     if error != nil {
                         print("Error occurred: \(String(describing: error))")
                     } else {
                         // Sign in successful.
                     }
                  })
        }
    }
}
```

Choose the run icon (
<problematic>|play|</problematic>
) in the top left of the Xcode window or type
<problematic>|Acommand|</problematic>
-R to build and run your app. You should see our pre-built sign-in UI for your app. Checkout the next steps to learn how to customize your UI (p. 179).

API References	• AWSMobileClient

	A library that initializes the SDK, fetches the AWS credentials, and creates a SDK SignInUI client instance. • Auth UserPools A wrapper Library for Amazon Cognito UserPools that provides a managed Email/Password sign-in UI. • Auth Core A library that caches and federates a login provider authentication token using Amazon Cognito Federated Identities, caches the federated AWS credentials, and handles the sign-in flow.

Setup Facebook Login in your Mobile App

Android - Java

Use Android API level 23 or higher	The AWS Mobile SDK library for Android sign-in (`aws-android-sdk-auth-ui`) provides the activity and view for presenting a `SignInUI` for the sign-in providers you configure. This library depends on the Android SDK API Level 23 or higher.

1. Add or update your AWS backend configuration file to incorporate your new sign-in. For details, see the last steps in the Get Started: Set Up Your Backend (p. 2) section.
2. Add the following permissions and Activity to your *AndroidManifest.xml* file:

```
<!-- ... -->

<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>

<!-- ... -->

<activity
    android:name="com.facebook.FacebookActivity"
    android:exported="true">
    <intent-filter>
        <action android:name="android.intent.action.VIEW" />
        <category android:name="android.intent.category.DEFAULT" />
        <category android:name="android.intent.category.BROWSABLE" />
        <data android:scheme="@string/fb_login_protocol_scheme" />
    </intent-filter>
</activity>

<!-- ... -->

<meta-data android:name="com.facebook.sdk.ApplicationId" android:value="@string/
facebook_app_id" />

<!-- ... -->
```

3. Add these dependencies to your *app/build.gradle* file:

```
dependencies {
  // Mobile Client for initializing the SDK
  implementation ('com.amazonaws:aws-android-sdk-mobile-client:2.6.+@aar')
 { transitive = true }

  // Facebook SignIn
  implementation 'com.android.support:support-v4:24.+'
  implementation ('com.amazonaws:aws-android-sdk-auth-facebook:2.6.+@aar')
 { transitive = true }

  // Sign in UI
  implementation 'com.android.support:appcompat-v7:24.+'
  implementation ('com.amazonaws:aws-android-sdk-auth-ui:2.6.+@aar') { transitive =
 true }
}
```

4. In `strings.xml`, add string definitions for your Facebook App ID and login protocol scheme.The value should contain your Facebook AppID in both cases, the login protocol value is always prefaced with `fb`.

```
<string name="facebook_app_id">1231231231232123123</string>
<string name="fb_login_protocol_scheme">fb1231231231232123123</string>
```

5. Create an activity that will present your sign-in screen.

   In Android Studio, choose **File > New > Activity > Basic Activity** and type an activity name, such as `AuthenticatorActivity`. If you want to make this your starting activity, move the the intent filter block containing `.LAUNCHER` to the `AuthenticatorActivity` in your app's `AndroidManifest.xml`.

```
<activity android:name=".AuthenticatorActivity">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

6. Update the `onCreate` function of your `AuthenticatorActivity` to call `AWSMobileClient`. This component provides the functionality to resume a signed-in authentication session. It makes a network call to retrieve the AWS credentials that allow users to access your AWS resources and registers a callback for when that transaction completes.

   If the user is signed in, the app goes to the `NextActivity`, otherwise it presents the user with the AWS Mobile ready made, configurable sign-in UI. `NextActivity` `Activity` class a user sees after a successful sign-in.

```
import android.app.Activity;
import android.os.Bundle;

import com.amazonaws.mobile.auth.ui.SignInUI;
import com.amazonaws.mobile.client.AWSMobileClient;
import com.amazonaws.mobile.client.AWSStartupHandler;
import com.amazonaws.mobile.client.AWSStartupResult;

public class AuthenticatorActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_authenticator);
```

```
        // Add a call to initialize AWSMobileClient
        AWSMobileClient.getInstance().initialize(this, new AWSStartupHandler() {
            @Override
            public void onComplete(AWSStartupResult awsStartupResult) {
                SignInUI signin = (SignInUI)
 AWSMobileClient.getInstance().getClient(AuthenticatorActivity.this, SignInUI.class);
                signin.login(AuthenticatorActivity.this,
 NextActivity.class).execute();
            }
        }).execute();
    }
}
```

Choose the run icon (
<problematic>|play|</problematic>
) in Android Studio to build your app and run it on your device/emulator. You should see our ready made sign-in UI for your app. Checkout the next steps to learn how to customize your UI (p. 179).

API References	• AWSMobileClient A library that initializes the SDK, constructs CredentialsProvider and AWSConfiguration objects, fetches the AWS credentials, and creates a SDK SignInUI client instance. • Auth UserPools A wrapper Library for Amazon Cognito UserPools that provides a managed Email/Password sign-in UI. • Auth Core A library that caches and federates a login provider authentication token using Amazon Cognito Federated Identities, caches the federated AWS credentials, and handles the sign-in flow.

Android - Kotlin

Use Android API level 23 or higher	The AWS Mobile SDK library for Android sign-in (`aws-android-sdk-auth-ui`) provides the activity and view for presenting a `SignInUI` for the sign-in providers you configure. This library depends on the Android SDK API Level 23 or higher.

1. Add or update your AWS backend configuration file to incorporate your new sign-in. For details, see the last steps in the Get Started: Set Up Your Backend (p. 2) section.
2. Add the following permissions and Activity to your *AndroidManifest.xml* file:

```
<!-- ... -->

<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>

<!-- ... -->
```

```
<activity
    android:name="com.facebook.FacebookActivity"
    android:exported="true">
    <intent-filter>
        <action android:name="android.intent.action.VIEW" />
        <category android:name="android.intent.category.DEFAULT" />
        <category android:name="android.intent.category.BROWSABLE" />
        <data android:scheme="@string/fb_login_protocol_scheme" />
    </intent-filter>
</activity>

<!-- ... -->

<meta-data android:name="com.facebook.sdk.ApplicationId" android:value="@string/
facebook_app_id" />

<!-- ... -->
```

3. Add these dependencies to your *app/build.gradle* file:

```
dependencies {
  // Mobile Client for initializing the SDK
  implementation ('com.amazonaws:aws-android-sdk-mobile-client:2.6.+@aar')
 { transitive = true }

  // Facebook SignIn
  implementation 'com.android.support:support-v4:24.+'
  implementation ('com.amazonaws:aws-android-sdk-auth-facebook:2.6.+@aar')
 { transitive = true }

  // Sign in UI
  implementation 'com.android.support:appcompat-v7:24.+'
  implementation ('com.amazonaws:aws-android-sdk-auth-ui:2.6.+@aar') { transitive =
 true }
}
```

4. In `strings.xml`, add string definitions for your Facebook App ID and login protocol scheme.The value should contain your Facebook AppID in both cases, the login protocol value is always prefaced with `fb`.

```
<string name="facebook_app_id">1231231231232123123</string>
<string name="fb_login_protocol_scheme">fb1231231231232123123</string>
```

5. Create an activity that will present your sign-in screen.

   In Android Studio, choose **File > New > Activity > Basic Activity** and type an activity name, such as `AuthenticatorActivity`. If you want to make this your starting activity, move the the intent filter block containing `.LAUNCHER` to the `AuthenticatorActivity` in your app's `AndroidManifest.xml`.

```
<activity android:name=".AuthenticatorActivity">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

6. Update the `onCreate` function of your `AuthenticatorActivity` to call `AWSMobileClient`. This component provides the functionality to resume a signed-in authentication session. It makes a network call to retrieve the AWS credentials that allow users to access your AWS resources and registers a callback for when that transaction completes.

If the user is signed in, the app goes to the `NextActivity`, otherwise it presents the user with the AWS Mobile ready made, configurable sign-in UI. `NextActivity` `Activity` class a user sees after a successful sign-in.

```
import android.app.Activity;
import android.os.Bundle;

import com.amazonaws.mobile.auth.ui.SignInUI;
import com.amazonaws.mobile.client.AWSMobileClient;
import com.amazonaws.mobile.client.AWSStartupHandler;
import com.amazonaws.mobile.client.AWSStartupResult;

class AuthenticatorActivity : Activity() {
  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)

    AWSMobileClient.getInstance().initialize(this) {
      val ui = AWSMobileClient.getInstance().getClient(this@AuthenticatorActivity,
 SignInUI::class.java)
      ui.login(this@AuthenticatorActivity, NextActivity::class.java).execute()
    }.execute()
  }
}
```

Choose the run icon (
<problematic>|play|</problematic>
) in Android Studio to build your app and run it on your device/emulator. You should see our ready made sign-in UI for your app. Checkout the next steps to learn how to customize your UI (p. 179).

API References	• AWSMobileClient A library that initializes the SDK, constructs CredentialsProvider and AWSConfiguration objects, fetches the AWS credentials, and creates a SDK SignInUI client instance. • Auth UserPools A wrapper Library for Amazon Cognito UserPools that provides a managed Email/Password sign-in UI. • Auth Core A library that caches and federates a login provider authentication token using Amazon Cognito Federated Identities, caches the federated AWS credentials, and handles the sign-in flow.

iOS - Swift

1. Add or update your AWS backend configuration file to incorporate your new sign-in. For details, see the last steps in the Get Started: Set Up Your Backend (p. 2) section.
2. Add the following dependencies in your project's `Podfile`.

```
platform :ios, '9.0'
  target :'YOUR-APP-NAME' do
    use_frameworks!
    pod 'AWSMobileClient', '~> 2.6.13'
```

```
    pod 'AWSFacebookSignIn', '~> 2.6.13'
    pod 'AWSAuthUI', '~> 2.6.13'
    # other pods
  end
```

Run `pod install --repo-update`.

If you encounter an error message that begins "`[!] Failed to connect to GitHub to update the CocoaPods/Specs . . .`", and your internet connectivity is working, you may need to update openssl and Ruby.

3. Add Facebook meta data to `Info.plist`.

   To configure your Xcode project to use Facebook Login, right-choose `Info.plist` and then choose **Open As > Source Code**.

   Add the following entry, using your project name, Facebook ID and login scheme ID.

```
<plist version="1.0">
<!-- ... -->
<dict>
<key>FacebookAppID</key>
<string>0123456789012345</string>
<key>FacebookDisplayName</key>
<string>YOUR-PROJECT-NAME</string>
<key>LSApplicationQueriesSchemes</key>
<array>
    <string>fbapi</string>
    <string>fb-messenger-api</string>
    <string>fbauth2</string>
    <string>fbshareextension</string>
</array>
<key>CFBundleURLTypes</key>
<array>
    <dict>
        <key>CFBundleURLSchemes</key>
        <array>
            <string>fb0123456789012345</string>
        </array>
    </dict>
</array>
</dict>
<!-- ... -->
```

4. Create a AWSMobileClient and initialize the SDK.

   Add code to create an instance of `AWSMobileClient` in the `application:open url` function of your `AppDelegate.swift`, to resume a previously signed-in authenticated session.

   Then add another instance of `AWSMobileClient` in the `didFinishLaunching` function to register the sign in providers, and to fetch an Amazon Cognito credentials that AWS will use to authorize access once the user signs in.

```
import UIKit

//import AWSMobileClient
import AWSMobileClient

@UIApplicationMain

class AppDelegate: UIResponder, UIApplicationDelegate {

    // Add a AWSMobileClient call in application:open url
```

```
    func application(_ application: UIApplication, open url: URL,
        sourceApplication: String?, annotation: Any) -> Bool {

        return AWSMobileClient.sharedInstance().interceptApplication(
            application, open: url,
            sourceApplication: sourceApplication,
            annotation: annotation)

    }

    // Add a AWSMobileClient call in application:didFinishLaunching
     func application(
        _ application: UIApplication,
            didFinishLaunchingWithOptions launchOptions:
                [UIApplicationLaunchOptionsKey: Any]?) -> Bool {

         return AWSMobileClient.sharedInstance().interceptApplication(
             application, didFinishLaunchingWithOptions:
             launchOptions)
    }

    // Other functions in AppDelegate . . .

  }
```

5. Implement your sign-in UI by calling the library provided by the SDK.

```
import UIKit
import AWSAuthCore
import AWSAuthUI

class SampleViewController: UIViewController {

    override func viewDidLoad() {

        super.viewDidLoad()

        if !AWSSignInManager.sharedInstance().isLoggedIn {
           AWSAuthUIViewController
             .presentViewController(with: self.navigationController!,
                  configuration: nil,
                  completionHandler: { (provider: AWSSignInProvider, error: Error?)
 in
                     if error != nil {
                         print("Error occurred: \(String(describing: error))")
                     } else {
                         // sign in successful.
                     }
                  })
        }
    }
}
```

Choose the run icon (
<problematic>|play|</problematic>
) in the top left of the Xcode window or type
<problematic>|Acommand|</problematic>
-R to build and run your app. You should see our pre-built sign-in UI for your app. Checkout the next steps to learn how to customize your UI (p. 179).

API References	• AWSMobileClient

	A library that initializes the SDK, fetches the AWS credentials, and creates a SDK SignInUI client instance. • Auth UserPools A wrapper Library for Amazon Cognito UserPools that provides a managed Email/Password sign-in UI. • Auth Core A library that caches and federates a login provider authentication token using Amazon Cognito Federated Identities, caches the federated AWS credentials, and handles the sign-in flow.

Setup Google Login in your Mobile App

Android - Java

Use Android API level 23 or higher	The AWS Mobile SDK library for Android sign-in (`aws-android-sdk-auth-ui`) provides the activity and view for presenting a `SignInUI` for the sign-in providers you configure. This library depends on the Android SDK API Level 23 or higher.

1. Add or update your AWS backend configuration file to incorporate your new sign-in. For details, see the last steps in the Get Started: Set Up Your Backend (p. 2) section.
2. Add these permissions to your *AndroidManifest.xml* file:

```
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
```

3. Add these dependencies to your *app/build.gradle* file:

```
dependencies {
    // Mobile Client for initializing the SDK
    implementation ('com.amazonaws:aws-android-sdk-mobile-client:2.6.+@aar')
 { transitive = true }

    // Google SignIn
    implementation 'com.android.support:support-v4:24.+'
    implementation ('com.amazonaws:aws-android-sdk-auth-google:2.6.+@aar')
 { transitive = true }

    // Sign in UI Library
    implementation 'com.android.support:appcompat-v7:24.+'
    implementation ('com.amazonaws:aws-android-sdk-auth-ui:2.6.+@aar') { transitive =
 true }
}
```

4. Create an activity that will present your sign-in screen.

   In Android Studio, choose **File > New > Activity > Basic Activity** and type an activity name, such as `AuthenticatorActivity`. If you want to make this your starting activity, move the the intent filter block containing `.LAUNCHER` to the `AuthenticatorActivity` in your app's `AndroidManifest.xml`.

```
<activity android:name=".AuthenticatorActivity">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

5. Update the `onCreate` function of your `AuthenticatorActivity` to call `AWSMobileClient`. This component provides the functionality to resume a signed-in authentication session. It makes a network call to retrieve the AWS credentials that allow users to access your AWS resources and registers a callback for when that transaction completes.

   If the user is signed in, the app goes to the `NextActivity`, otherwise it presents the user with the AWS Mobile ready made, configurable sign-in UI. `NextActivity` `Activity` class a user sees after a successful sign-in.

```
import android.app.Activity;
import android.os.Bundle;

import com.amazonaws.mobile.auth.ui.SignInUI;
import com.amazonaws.mobile.client.AWSMobileClient;
import com.amazonaws.mobile.client.AWSStartupHandler;
import com.amazonaws.mobile.client.AWSStartupResult;

public class AuthenticatorActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_authenticator);

        // Add a call to initialize AWSMobileClient
        AWSMobileClient.getInstance().initialize(this, new AWSStartupHandler() {
            @Override
            public void onComplete(AWSStartupResult awsStartupResult) {
                SignInUI signin = (SignInUI)
 AWSMobileClient.getInstance().getClient(AuthenticatorActivity.this, SignInUI.class);
                signin.login(AuthenticatorActivity.this,
 MainActivity.class).execute();
            }
        }).execute();
    }
}
```

Choose the run icon (
<problematic>|play|</problematic>
) in Android Studio to build your app and run it on your device/emulator. You should see our ready made sign-in UI for your app. Checkout the next steps to learn how to customize your UI (p. 179).

API References	• AWSMobileClient A library that initializes the SDK, constructs CredentialsProvider and AWSConfiguration objects, fetches the AWS credentials, and creates a SDK SignInUI client instance. • Auth UserPools A wrapper Library for Amazon Cognito UserPools that provides a managed Email/Password sign-in UI.

	• Auth Core A library that caches and federates a login provider authentication token using Amazon Cognito Federated Identities, caches the federated AWS credentials, and handles the sign-in flow.

Android - Kotlin

Use Android API level 23 or higher	The AWS Mobile SDK library for Android sign-in (`aws-android-sdk-auth-ui`) provides the activity and view for presenting a `SignInUI` for the sign-in providers you configure. This library depends on the Android SDK API Level 23 or higher.

1. Add or update your AWS backend configuration file to incorporate your new sign-in. For details, see the last steps in the Get Started: Set Up Your Backend (p. 2) section.
2. Add these permissions to your *AndroidManifest.xml* file:

```
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
```

3. Add these dependencies to your *app/build.gradle* file:

```
dependencies {
    // Mobile Client for initializing the SDK
    implementation ('com.amazonaws:aws-android-sdk-mobile-client:2.6.+@aar')
 { transitive = true }

    // Google SignIn
    implementation 'com.android.support:support-v4:24.+'
    implementation ('com.amazonaws:aws-android-sdk-auth-google:2.6.+@aar')
 { transitive = true }

    // Sign in UI Library
    implementation 'com.android.support:appcompat-v7:24.+'
    implementation ('com.amazonaws:aws-android-sdk-auth-ui:2.6.+@aar') { transitive =
 true }
}
```

4. Create an activity that will present your sign-in screen.

   In Android Studio, choose **File > New > Activity > Basic Activity** and type an activity name, such as `AuthenticatorActivity`. If you want to make this your starting activity, move the the intent filter block containing `.LAUNCHER` to the `AuthenticatorActivity` in your app's `AndroidManifest.xml`.

```
<activity android:name=".AuthenticatorActivity">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

5. Update the `onCreate` function of your `AuthenticatorActivity` to call `AWSMobileClient`. This component provides the functionality to resume a signed-in authentication session. It makes

a network call to retrieve the AWS credentials that allow users to access your AWS resources and registers a callback for when that transaction completes.

If the user is signed in, the app goes to the `NextActivity`, otherwise it presents the user with the AWS Mobile ready made, configurable sign-in UI. `NextActivity` `Activity` class a user sees after a successful sign-in.

```
import android.app.Activity;
import android.os.Bundle;

import com.amazonaws.mobile.auth.ui.SignInUI;
import com.amazonaws.mobile.client.AWSMobileClient;
import com.amazonaws.mobile.client.AWSStartupHandler;
import com.amazonaws.mobile.client.AWSStartupResult;

class AuthenticatorActivity : Activity() {
  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)

    AWSMobileClient.getInstance().initialize(this) {
      val ui = AWSMobileClient.getInstance().getClient(this@AuthenticatorActivity,
 SignInUI::class.java)
      ui.login(this@AuthenticatorActivity, NextActivity::class.java).execute()
    }.execute()
  }
}
```

Choose the run icon (
<problematic>|play|</problematic>
) in Android Studio to build your app and run it on your device/emulator. You should see our ready made sign-in UI for your app. Checkout the next steps to learn how to customize your UI (p. 179).

API References	• AWSMobileClient A library that initializes the SDK, constructs CredentialsProvider and AWSConfiguration objects, fetches the AWS credentials, and creates a SDK SignInUI client instance. • Auth UserPools A wrapper Library for Amazon Cognito UserPools that provides a managed Email/Password sign-in UI. • Auth Core A library that caches and federates a login provider authentication token using Amazon Cognito Federated Identities, caches the federated AWS credentials, and handles the sign-in flow.

iOS - Swift

1. Add or update your AWS backend configuration file to incorporate your new sign-in. For details, see the last steps in the Get Started: Set Up Your Backend (p. 2) section.
2. Add the following dependencies in the Podfile.

```
platform :ios, '9.0'
```

```
  target :'YOUR-APP-NAME' do
    use_frameworks!
    pod 'AWSMobileClient', '~> 2.6.13'
    pod 'AWSGoogleSignIn', '~> 2.6.13'
    pod 'AWSAuthUI', '~> 2.6.13'
    pod 'GoogleSignIn', '~> 4.0'
    # other pods
  end
```

Run `pod install --repo-update` before you continue.

If you encounter an error message that begins "`[!] Failed to connect to GitHub to update the CocoaPods/Specs . . .`", and your internet connectivity is working, you may need to update openssl and Ruby.

3. Add Google metadata to info.plist

   To configure your Xcode project to use Google Login, open its Info.plist file using **Right-click > Open As > Source Code.** Add the following entry. Substitute your project name for the placeholder string.

```
<plist version="1.0">
<!-- ... -->
<key>CFBundleURLTypes</key>
<array>
    <dict>
    <key>CFBundleURLSchemes</key>
    <array>
        <string>com.googleusercontent.apps.xxxxxxxxxxxx-
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx</string>
    </array>
    </dict>
</array>
<!-- ... -->
```

4. Create a AWSMobileClient and initialize the SDK.

   Add code to create an instance of `AWSMobileClient` in the `application:open url` function of your `AppDelegate.swift`, to resume a previously signed-in authenticated session.

   Then add another instance of `AWSMobileClient` in the `didFinishLaunching` function to register the sign in providers, and to fetch an Amazon Cognito credentials that AWS will use to authorize access once the user signs in.

```
import UIKit

//import AWSMobileClient
import AWSMobileClient

@UIApplicationMain

class AppDelegate: UIResponder, UIApplicationDelegate {

    // Add a AWSMobileClient call in application:open url
    func application(_ application: UIApplication, open url: URL,
        sourceApplication: String?, annotation: Any) -> Bool {

        return AWSMobileClient.sharedInstance().interceptApplication(
            application, open: url,
            sourceApplication: sourceApplication,
            annotation: annotation)

    }
```

```
    // Add a AWSMobileClient call in application:didFinishLaunching
    func application(
        _ application: UIApplication,
            didFinishLaunchingWithOptions launchOptions:
                [UIApplicationLaunchOptionsKey: Any]?) -> Bool {

         return AWSMobileClient.sharedInstance().interceptApplication(
             application, didFinishLaunchingWithOptions:
             launchOptions)
    }

    // Other functions in AppDelegate . . .

  }
```

5. Implement your sign-in UI by calling the library provided by the SDK.

```
import UIKit
import AWSAuthCore
import AWSAuthUI

class SampleViewController: UIViewController {

    override func viewDidLoad() {

        super.viewDidLoad()

        if !AWSSignInManager.sharedInstance().isLoggedIn {
           AWSAuthUIViewController
             .presentViewController(with: self.navigationController!,
                  configuration: nil,
                  completionHandler: { (provider: AWSSignInProvider, error: Error?)
 in
                     if error != nil {
                         print("Error occurred: \(String(describing: error))")
                     } else {
                         // Sign in successful.
                     }
                  })
        }
    }
}
```

Choose the run icon (
<problematic>|play|</problematic>
) in the top left of the Xcode window or type
<problematic>|Acommand|</problematic>
-R to build and run your app. You should see our pre-built sign-in UI for your app. Checkout the next steps to learn how to customize your UI (p. 179).

API References	• AWSMobileClient A library that initializes the SDK, fetches the AWS credentials, and creates a SDK SignInUI client instance. • Auth UserPools A wrapper Library for Amazon Cognito UserPools that provides a managed Email/Password sign-in UI. • Auth Core

	A library that caches and federates a login provider authentication token using Amazon Cognito Federated Identities, caches the federated AWS credentials, and handles the sign-in flow.

Enable Sign-out

Android - Java

To enable a user to sign-out of your app, register a callback for sign-in events by adding a `SignInStateChangeListener` to `IdentityManager`. The listener captures both `onUserSignedIn` and `onUserSignedOut` events.

```
IdentityManager.getDefaultIdentityManager().addSignInStateChangeListener(new
 SignInStateChangeListener() {
    @Override
    // Sign-in listener
    public void onUserSignedIn() {
        Log.d(LOG_TAG, "User Signed In");
    }

    // Sign-out listener
    @Override
    public void onUserSignedOut() {

        // return to the sign-in screen upon sign-out
       showSignIn();
    }
});
```

To initiate a sign-out, call the `signOut` method of `IdentityManager`.

```
IdentityManager.getDefaultIdentityManager().signOut();
```

Android - Kotlin

To enable a user to sign-out of your app, register a callback for sign-in events by adding a `SignInStateChangeListener` to `IdentityManager`. The listener captures both `onUserSignedIn` and `onUserSignedOut` events.

```
IdentityManager.getDefaultIdentityManager().addSignInStateChangeListener(
    object : SignInStateChangeListener() {
        override fun onUserSignedIn() {
            Log.d(TAG, "User signed in");
        }

        override fun onUserSignedOut() {
            Log.d(TAG, "User signed out");
        }
    }
);
```

To initiate a sign-out, call the `signOut` method of `IdentityManager`.

```
IdentityManager.getDefaultIdentityManager().signOut();
```

iOS - Swift

To initiate a sign-out, add a call to `AWSSignInManager.sharedInstance().logout`.

```
@IBAction func signOutButtonPress(_ sender: Any) {

    AWSSignInManager.sharedInstance().logout(completionHandler: {(result: Any?, error:
 Error?) in
        self.showSignIn()
        // print("Sign-out Successful: \(signInProvider.getDisplayName)");

    })
}
```

For a fuller example, see Sign-out a Signed-in User (p. 153) in the How To section.

Next Steps

- Customize the UI (p. 179)
- Import Your Exisiting Amazon Cognito Identity Pool (p. 147)
- Amazon Cognito Developer Guide

Add Push Notifications to Your Mobile App with Amazon Pinpoint

Overview

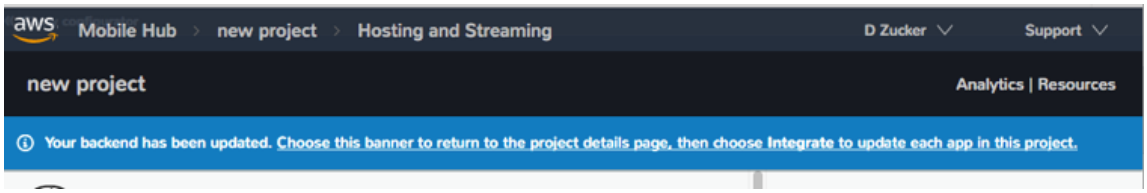
Mobile Hub deploys your Push Notifications backend services when you enable the Messaging and Analytics (p. 340) feature using the Amazon Pinpoint service. Amazon Pinpoint enables apps to receive mobile push messages sent from the Apple (APNs) and Google (FCM/GCM) platforms. You can also create Amazon Pinpoint campaigns that tie user behavior to push or other forms of messaging.

Set Up Your Backend

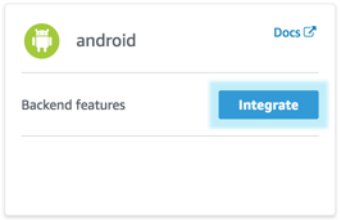
1. Complete the Get Started (p. 2) steps before your proceed.
2. Choose the **Messaging and Analytics** tile
3. Choose **Mobile push**.

   For Android - Firebase/Google Cloud Messaging (FCM/GCM): Choose **Android** and provide your Firebase/Google application API key and Sender ID. To retrieve or create these values, see Setting Up Android Push Notifications .

   For iOS - Apple Push Notification Service (APNs): Choose **iOS** and provide your Apple app P12 Certificate and, optionally, Certificate password. To retrieve or create these items, see Setting Up iOS Push Notifications.
4. When the operation is complete, an alert will pop up saying "Your Backend has been updated", prompting you to download the latest copy of the cloud configuration file. If you're done with configuring the feature, choose the banner to return to the project details page.



5. From the project detail page, every app that needs to be updated with the latest cloud configuration file will have a flashing **Integrate** button. Choose the button to enter the integrate wizard.



6. Update your app with the latest copy of the cloud configuration file. Your app now references the latest version of your backend. Choose Next and follow the Push Notification documentation below to connect to your backend.

Connect to your backend

To add push notification to your app

Android - Java

1. Set up AWS Mobile SDK components with the following Set Up Your Backend (p. 2) steps.
   a. `AndroidManifest.xml` must contain:

```
<uses-permission android:name="android.permission.WAKE_LOCK"/>
<uses-permission android:name="com.google.android.c2dm.permission.RECEIVE" />
<permission android:name="com.mysampleapp.permission.C2D_MESSAGE"
            android:protectionLevel="signature" />
<uses-permission android:name="com.mysampleapp.permission.C2D_MESSAGE" />

<application

    <!--Add these to your Application declaration
    to filter for the notification intent-->
    <receiver
        android:name="com.google.android.gms.gcm.GcmReceiver"
        android:exported="true"
        android:permission="com.google.android.c2dm.permission.SEND" >
        <intent-filter>
            <action android:name="com.google.android.c2dm.intent.RECEIVE" />
            <category android:name="com.mysampleapp" />
        </intent-filter>
    </receiver>

    <service
        android:name=".PushListenerService"
        android:exported="false" >
        <intent-filter>
            <action android:name="com.google.android.c2dm.intent.RECEIVE" />
        </intent-filter>
    </service>

</application>
```

b. Add the following to your `app/build.gradle`:

```
dependencies{
    implementation 'com.amazonaws:aws-android-sdk-pinpoint:2.6.+'
    implementation ('com.amazonaws:aws-android-sdk-auth-core:2.6.+@aar')
 {transitive = true;}

    implementation 'com.google.android.gms:play-services-iid:11.6.0'
    implementation 'com.google.android.gms:play-services-gcm:11.6.0'
}
```

c. Add the following to your project level `build.gradle`:

```
buildscript {
    dependencies {
        classpath 'com.google.gms:google-services:3.1.1'
    }
}

allprojects {
    repositories {
        maven {
            url "https://maven.google.com"
        }
    }
}
```

2. Create an Amazon Pinpoint client in the location of your push notification code.

```
import com.amazonaws.mobileconnectors.pinpoint.PinpointConfiguration;
import com.amazonaws.mobileconnectors.pinpoint.PinpointManager;
import com.google.android.gms.gcm.GoogleCloudMessaging;
import com.google.android.gms.iid.InstanceID;

public class MainActivity extends AppCompatActivity {
     public static final String LOG_TAG = MainActivity.class.getSimpleName();

     public static PinpointManager pinpointManager;

     @Override
     protected void onCreate(Bundle savedInstanceState) {
         super.onCreate(savedInstanceState);
         setContentView(R.layout.activity_main);

         if (pinpointManager == null) {
             PinpointConfiguration pinpointConfig = new PinpointConfiguration(
                     getApplicationContext(),
                     AWSMobileClient.getInstance().getCredentialsProvider(),
                     AWSMobileClient.getInstance().getConfiguration());

             pinpointManager = new PinpointManager(pinpointConfig);

             new Thread(new Runnable() {
                 @Override
                 public void run() {
                   try {
                       String deviceToken =
                         InstanceID.getInstance(MainActivity.this).getToken(
                             "123456789Your_GCM_Sender_Id",
                             GoogleCloudMessaging.INSTANCE_ID_SCOPE);
                       Log.e("NotError", deviceToken);
                       pinpointManager.getNotificationClient()
                                      .registerGCMDeviceToken(deviceToken);
```

```
                } catch (Exception e) {
                    e.printStackTrace();
                }
                }
            }).start();
        }
    }
}
```

Android - Kotlin

1. Set up AWS Mobile SDK components with the following Set Up Your Backend (p. 2) steps.

   a. `AndroidManifest.xml` must contain:

```
<uses-permission android:name="android.permission.WAKE_LOCK"/>
<uses-permission android:name="com.google.android.c2dm.permission.RECEIVE" />
<permission android:name="com.mysampleapp.permission.C2D_MESSAGE"
            android:protectionLevel="signature" />
<uses-permission android:name="com.mysampleapp.permission.C2D_MESSAGE" />

<application

    <!--Add these to your Application declaration
    to filter for the notification intent-->
    <receiver
        android:name="com.google.android.gms.gcm.GcmReceiver"
        android:exported="true"
        android:permission="com.google.android.c2dm.permission.SEND" >
        <intent-filter>
            <action android:name="com.google.android.c2dm.intent.RECEIVE" />
            <category android:name="com.mysampleapp" />
        </intent-filter>
    </receiver>

    <service
        android:name=".PushListenerService"
        android:exported="false" >
        <intent-filter>
            <action android:name="com.google.android.c2dm.intent.RECEIVE" />
        </intent-filter>
    </service>

</application>
```

   b. Add the following to your `app/build.gradle`:

```
dependencies{
    implementation 'com.amazonaws:aws-android-sdk-pinpoint:2.6.+'
    implementation ('com.amazonaws:aws-android-sdk-auth-core:2.6.+@aar')
 {transitive = true;}

    implementation 'com.google.android.gms:play-services-iid:11.6.0'
    implementation 'com.google.android.gms:play-services-gcm:11.6.0'
}
```

   c. Add the following to your project level `build.gradle`:

```
buildscript {
    dependencies {
        classpath 'com.google.gms:google-services:3.1.1'
    }
```

```
}

allprojects {
    repositories {
        maven {
            url "https://maven.google.com"
        }
    }
}
```

2. Create an Amazon Pinpoint client in the location of your push notification code.

```
import com.amazonaws.mobileconnectors.pinpoint.PinpointConfiguration;
import com.amazonaws.mobileconnectors.pinpoint.PinpointManager;
import com.google.android.gms.gcm.GoogleCloudMessaging;
import com.google.android.gms.iid.InstanceID;

class MainActivity : AppCompatActivity() {
    companion object {
        private val LOG_TAG = this::class.java.getSimpleName
        var pinpointManager: PinpointManager? = null
    }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        AWSMobileClient.getInstance().initialize(this).execute()
        with (AWSMobileClient.getInstance()) {
            if (pinpointManager == null) {
                val config = PinpointConfiguration(applicationContext,
 credentialsProvider, configuration)
                pinpointManager = PinpointManager(config)
            }
        }

        thread(start = true) {
            try {
                val deviceToken = InstanceID.getInstance(this@MainActivity)
                    .getToken("YOUR-GCM-SENDER-ID",
 GoogleCloudMessaging.INSTANCE_ID_SCOPE)
                Log.i(LOG_TAG, "GCM DeviceToken = $deviceToken")

 pinpointManager?.notificationClient?.registerGCMDeviceToken(deviceToken)
            } catch (e: Exception) {
                e.printStackTrace()
            }
        }
    }
}
```

iOS - Swift

1. Set up AWS Mobile SDK components with the following Set Up Your Backend (p. 2) steps.
   a. `Podfile` that you configure to install the AWS Mobile SDK must contain:

```
platform :ios, '9.0'

target :'YOUR-APP-NAME' do
  use_frameworks!

    pod  'AWSPinpoint', '~> 2.6.13'
```

```
    # other pods

end
```

Run `pod install --repo-update` before you continue.

If you encounter an error message that begins "`[!] Failed to connect to GitHub to update the CocoaPods/Specs . . .`", and your internet connectivity is working, you may need to update openssl and Ruby.

b. Classes that call Amazon Pinpoint APIs must use the following import statements:

```
import AWSCore
import AWSPinpoint
```

2. Create an Amazon Pinpoint client by using the following code into the `didFinishLaunchwithOptions` method of your app's `AppDelegate.swift`. This will also register your device token with Amazon Pinpoint.

```
var pinpoint: AWSPinpoint?


func application(_ application: UIApplication, didFinishLaunchingWithOptions
 launchOptions:
    [UIApplicationLaunchOptionsKey: Any]?) -> Bool {

    pinpoint =
        AWSPinpoint(configuration:
            AWSPinpointConfiguration.defaultPinpointConfiguration(launchOptions:
 launchOptions))

    return true
}
```

Add Amazon Pinpoint Targeted and Campaign Push Messaging

Amazon Pinpoint console enables you to target your app users with push messaging. You can send individual messages or configure campaigns that target a group of users that match a profile that you define. For instance, you could email users that have not used the app in 30 days, or send an SMS to those that frequently use a given feature of your app.

Android - Java

The following 2 steps show how to receive push notifications targeted for your app.

1. Add a Push Listener Service to Your App.

   The name of the class must match the push listener service name used in the app manifest. `pinpointManager` is a reference to the static PinpointManager variable declared in the MainActivity shown in a previous step. Use the following steps to set up Push Notification listening in your app.

   a. The following push listener code assumes that the app's MainActivity is configured using the manifest setup described in a previous section.

```
import android.content.Intent;
import android.os.Bundle;
import android.support.v4.content.LocalBroadcastManager;
import android.util.Log;
```

```
import
 com.amazonaws.mobileconnectors.pinpoint.targeting.notification.NotificationClient;
import com.google.android.gms.gcm.GcmListenerService;

public class YOUR-PUSH-LISTENER-SERVICE-NAME extends GcmListenerService {
    public static final String LOGTAG = PushListenerService.class.getSimpleName();

    // Intent action used in local broadcast
    public static final String ACTION_PUSH_NOTIFICATION = "push-notification";
    // Intent keys
    public static final String INTENT_SNS_NOTIFICATION_FROM = "from";
    public static final String INTENT_SNS_NOTIFICATION_DATA = "data";

    /**
     * Helper method to extract push message from bundle.
     *
     * @param data bundle
     * @return message string from push notification
     */
    public static String getMessage(Bundle data) {
        // If a push notification is sent as plain
        // text, then the message appears in "default".
        // Otherwise it's in the "message" for JSON format.
        return data.containsKey("default") ? data.getString("default") :
 data.getString(
                "message", "");
    }

    private void broadcast(final String from, final Bundle data) {
        Intent intent = new Intent(ACTION_PUSH_NOTIFICATION);
        intent.putExtra(INTENT_SNS_NOTIFICATION_FROM, from);
        intent.putExtra(INTENT_SNS_NOTIFICATION_DATA, data);
        LocalBroadcastManager.getInstance(this).sendBroadcast(intent);
    }

    @Override
    public void onMessageReceived(final String from, final Bundle data) {
        Log.d(LOGTAG, "From:" + from);
        Log.d(LOGTAG, "Data:" + data.toString());

        final NotificationClient notificationClient =
            MainActivity.pinpointManager.getNotificationClient();

        NotificationClient.CampaignPushResult pushResult =
                notificationClient.handleGCMCampaignPush(from, data,
 this.getClass());

        if (!NotificationClient.CampaignPushResult.NOT_HANDLED.equals(pushResult))
 {
            // The push message was due to a Pinpoint campaign.
            // If the app was in the background, a local notification was added
            // in the notification center. If the app was in the foreground, an
            // event was recorded indicating the app was in the foreground,
            // for the demo, we will broadcast the notification to let the main
            // activity display it in a dialog.
            if (

 NotificationClient.CampaignPushResult.APP_IN_FOREGROUND.equals(pushResult)) {
                    // Create a message that will display the raw
                    //data of the campaign push in a dialog.
                    data.putString("
                        message",
                        String.format("Received Campaign Push:\n%s",
 data.toString()));
                    broadcast(from, data);
            }
```

```
            return;
        }
    }
}
```

b. Add code to react to your push listener service.

The following code can be placed where your app will react to incoming notifications.

```
import android.app.Activity;
import android.app.AlertDialog;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.support.v4.content.LocalBroadcastManager;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;

public class MainActivity extends AppCompatActivity {
    public static final String LOG_TAG = MainActivity.class.getSimpleName();

    @Override
    protected void onPause() {
        super.onPause();

        // unregister notification receiver

 LocalBroadcastManager.getInstance(this).unregisterReceiver(notificationReceiver);
    }

    @Override
    protected void onResume() {
        super.onResume();

        // register notification receiver

 LocalBroadcastManager.getInstance(this).registerReceiver(notificationReceiver,
                new IntentFilter(PushListenerService.ACTION_PUSH_NOTIFICATION));
    }

    private final BroadcastReceiver notificationReceiver = new BroadcastReceiver()
 {
        @Override
        public void onReceive(Context context, Intent intent) {
            Log.d(LOG_TAG, "Received notification from local broadcast. Display it
 in a dialog.");

            Bundle data =
 intent.getBundleExtra(PushListenerService.INTENT_SNS_NOTIFICATION_DATA);
            String message = PushListenerService.getMessage(data);

            new AlertDialog.Builder(MainActivity.this)
                    .setTitle("Push notification")
                    .setMessage(message)
                    .setPositiveButton(android.R.string.ok, null)
                    .show();
        }
    };

}
```

Android - Kotlin

The following 2 steps show how to receive push notifications targeted for your app.

1. Add a Push Listener Service to Your App.

   The name of the class must match the push listener service name used in the app manifest. `pinpointManager` is a reference to the static PinpointManager variable declared in the MainActivity shown in a previous step. Use the following steps to set up Push Notification listening in your app.

   a. The following push listener code assumes that the app's MainActivity is configured using the manifest setup described in a previous section.

```
import android.content.Intent;
import android.os.Bundle;
import android.support.v4.content.LocalBroadcastManager;
import android.util.Log;

import
 com.amazonaws.mobileconnectors.pinpoint.targeting.notification.NotificationClient;
import com.google.android.gms.gcm.GcmListenerService;

class YOUR-PUSH-LISTENER-SERVICE-NAME : GcmListenerService() {
    companion object {
        private val LOG_TAG = this::class.java.simpleName
        const val ACTION_PUSH_NOTIFICATION: String = "push-notification"
        const val INTENT_SNS_NOTIFICATION_FROM: String = "from"
        const val INTENT_SNS_NOTIFICATION_DATA: String = "data"

        // Helper method to extract push message from bundle.
        fun getMessage(data: Bundle) =
            if (data.containsKey("default")
                data.getString("default")
            else
                data.getString("message", "")
    }

    private fun broadcast(from: String, data: Bundle) {
        val intent = Intent(ACTION_PUSH_NOTIFICATION).apply {
            putExtra(INTENT_SNS_NOTIFICATION_FROM, from)
            putExtra(INTENT_SNS_NOTIFICATION_DATA, data)
        }
        LocalBroadcastManager.getInstance(this).sendBroadcast(intent)
    }

    override fun onMessageReceived(from: String?, data: Bundle?) {
        Log.d(LOG_TAG, "From: $from")
        Log.d(LOG_TAG, "Data: $data")

        val notificationClient =
 MainActivity.pinpointManager!!.notificationClient!!
        val pushResult = notificationClient.handleGCMCampaignPush(from, data,
 this::class.java)
        if (pushResult != NotificationClient.CampaignPushResult.NOT_HANDLED) {
            // The push message was due to a Pinpoint campaign
            // If the app was in the background, a local notification was added
            // in the notification center. If the app was in the foreground, an
            // event was recorded indicating the app was in the foreground,
            // for the demo, we will broadcast the notification to let the main
            // activity display it in a dialog.
            if (pushResult ==
 NotificationClient.CampaignPushResult.APP_IN_FOREGROUND) {
                data.putString("message", "Received Campaign Push:\n$data")
                broadcast(from, data)
```

```
            }
            return
        }
    }
}
```

b. Add code to react to your push listener service.

The following code can be placed where your app will react to incoming notifications.

```
import android.app.Activity;
import android.app.AlertDialog;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.support.v4.content.LocalBroadcastManager;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;

class MainActivity : AppCompatActivity() {
    companion object {
        // ...

        val notificationReceiver = object : BroadcastReceiver() {
            override fun onReceive(context: Context, intent: Intent) {
                Log.d(LOG_TAG, "Received notification from local broadcast.")

                val data =
 intent.getBundleExtra(PushListenerService.INTENT_SNS_NOTIFICATION_DATA)
                val message = PushListenerService.getMessage(data)

                // Uses anko library to display an alert dialog
                alert(message) {
                    title = "Push notification"
                    positiveButton("OK") { /* Do nothing */ }
                }.show()
            }
        }
    }

    override fun onPause() {
        super.onPause()

 LocalBroadcastManager.getInstance(this).unregisterReceiver(notificationReceiver)
    }

    override fun onResume() {
        super.onResume()

 LoadBroadcastManager.getInstance(this).registerReceiver(notificationReceiver,
            IntentFilter(PushListenerService.ACTION_PUSH_NOTIFICATION))
    }

    // ...
}
```

iOS - Swift

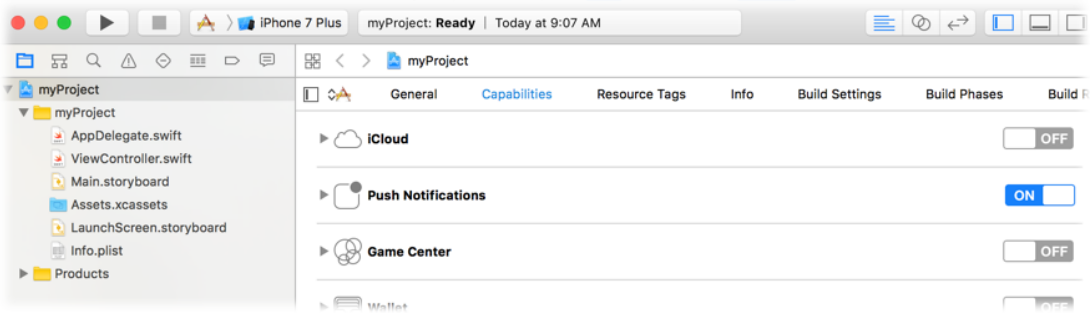
1. In your `AppDelegate` with `PinpointManager` instantiated, make sure the push listening code exists in the following functions.

```
// . . .

    func application(
        _ application: UIApplication,
                       didRegisterForRemoteNotificationsWithDeviceToken deviceToken:
 Data) {

            pinpoint!.notificationManager.interceptDidRegisterForRemoteNotifications(
                    withDeviceToken: deviceToken)
    }

    func application(
        _ application: UIApplication,
                       didReceiveRemoteNotification userInfo: [AnyHashable: Any],
                       fetchCompletionHandler completionHandler:
                            @escaping (UIBackgroundFetchResult) -> Void) {

            pinpoint!.notificationManager.interceptDidReceiveRemoteNotification(
                    userInfo, fetchCompletionHandler: completionHandler)

        if (application.applicationState == .active) {
            let alert = UIAlertController(title: "Notification Received",
                                          message: userInfo.description,
                                          preferredStyle: .alert)
            alert.addAction(UIAlertAction(title: "Ok", style: .default, handler:
 nil))

            UIApplication.shared.keyWindow?.rootViewController?.present(
                alert, animated: true, completion:nil)
        }
    }
// . . .
}
```

2. Add the following code in the `ViewController` where you request notification permissions.

```
var userNotificationTypes : UIUserNotificationType
userNotificationTypes = [.alert , .badge , .sound]
let notificationSettings = UIUserNotificationSettings.init(types:
 userNotificationTypes, categories: nil)
UIApplication.shared.registerUserNotificationSettings(notificationSettings)
UIApplication.shared.registerForRemoteNotifications()
```

3. In Xcode, choose your app target in the Project Navigator, choose **Capabilities**, turn on **Push Notifications**.



4. Build and run your app using information at Building the Sample iOS App From AWS Mobile Hub.

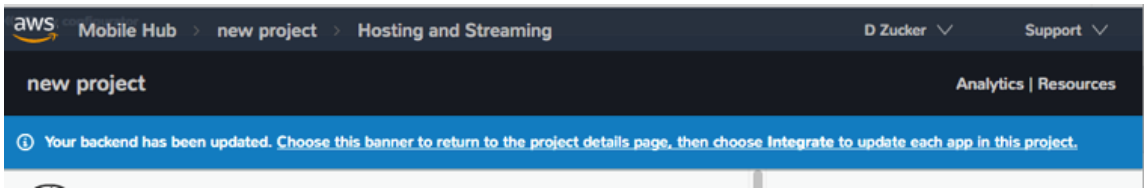
Add NoSQL Database to Your Mobile App with Amazon DynamoDB

Overview

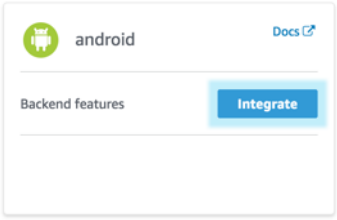
The AWS Mobile Hub nosqldb feature uses Amazon DynamoDB to enable you to create database tables that can store and retrieve data for use by your apps.

Set Up Your Backend

1. Complete the Get Started (p. 2) steps before you proceed.
2. Enable **NoSQL Database**: Open your project in Mobile Hub and choose the **NoSQL Database** tile to enable the feature.
3. Follow the console work flow to define the tables you need. See Configuring the NoSQL Database Feature (p. 336) for details.
4. When the operation is complete, an alert will pop up saying "Your Backend has been updated", prompting you to download the latest copy of the cloud configuration file. If you're done configuring the feature, choose the banner to return to the project details page.



5. From the project detail page, every app that needs to be updated with the latest cloud configuration file will have a flashing **Integrate** button. Choose the button to enter the integrate wizard.



6. Update your app with the latest copy of the cloud configuration file. Your app now references the latest version of your backend. Choose Next and follow the NoSQL Database documentation below to connect to your backend.
7. Download the models required for your app. The data models provide set and get methods for each attribute of a DynamoDB table.

Connect to your backend

To add AWS Mobile NoSQL Database to your app

Android - Java

1. Set up AWS Mobile SDK components with the following Set Up Your Backend (p. 2) steps.
   a. `app/build.gradle` must contain:

```
dependencies{
    implementation 'com.amazonaws:aws-android-sdk-ddb-mapper:2.6.+'
```

```
}
```

b. For each Activity where you make calls to perform database operations, import the following APIs.

```
import com.amazonaws.mobileconnectors.dynamodbv2.dynamodbmapper.DynamoDBMapper;
```

2. Create a `DynamoDBMapper` client for your app as in the following example.

```
import com.amazonaws.auth.AWSCredentialsProvider;
import com.amazonaws.mobile.client.AWSMobileClient;
import com.amazonaws.mobile.config.AWSConfiguration;

import com.amazonaws.mobileconnectors.dynamodbv2.dynamodbmapper.DynamoDBMapper;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClient;

import java.util.Random;

public class MainActivity extends AppCompatActivity {

    // Declare a DynamoDBMapper object
    DynamoDBMapper dynamoDBMapper;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        // AWSMobileClient enables AWS user credentials to access your table
        AWSMobileClient.getInstance().initialize(this).execute();

        AWSCredentialsProvider credentialsProvider =
 AWSMobileClient.getInstance().getCredentialsProvider();
        AWSConfiguration configuration =
 AWSMobileClient.getInstance().getConfiguration();


        // Add code to instantiate a AmazonDynamoDBClient
        AmazonDynamoDBClient dynamoDBClient = new
 AmazonDynamoDBClient(credentialsProvider);

        this.dynamoDBMapper = DynamoDBMapper.builder()
                .dynamoDBClient(dynamoDBClient)
                .awsConfiguration(configuration)
                .build();

        // other activity code ...
    }
}
```

3. Add the project data model files you downloaded from the Mobile Hub console. The data models provide set and get methods for each attribute of a DynamoDB table they model.

a. Copy the data model file(s) you downloaded, `./YOUR-PROJECT-NAME-integration-lib-aws-my-sample-app-android/src/main/java/com/amazonaws/models/nosqlYOUR-TABLE-NAMEDO.java` into the Android Studio folder that contains your main activity.

Note
Use Asynchronous Calls to DynamoDB

Since calls to DynamoDB are synchronous, they don't belong on your UI thread. Use an asynchronous method like the `Runnable` wrapper to call `DynamoDBObjectMapper` in a separate thread.

```
Runnable runnable = new Runnable() {
     public void run() {
     //DynamoDB calls go here
     }
};
Thread mythread = new Thread(runnable);
mythread.start();
```

Android - Kotlin

1. Set up AWS Mobile SDK components with the following Set Up Your Backend (p. 2) steps.
   a. `app/build.gradle` must contain:

   ```
   dependencies{
       implementation 'com.amazonaws:aws-android-sdk-ddb-mapper:2.6.+'
   }
   ```

   b. For each Activity where you make calls to perform database operations, import the following APIs.

   ```
   import com.amazonaws.mobileconnectors.dynamodbv2.dynamodbmapper.DynamoDBMapper;
   ```

2. Create a `DynamoDBMapper` client for your app as in the following example.

```
// import DynamoDBMapper
import com.amazonaws.mobileconnectors.dynamodbv2.dynamodbmapper.DynamoDBMapper;

class MainActivity : AppCompatActivity() {
  private var dynamoDBMapper: DynamoDBMapper? = null

  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)

    val client =
AmazonDynamoDBClient(AWSMobileClient.getInstance().credentialsProvider)
    dynamoDBMapper = DynamoDBMapper.builder()
        .dynamoDBClient(client)
        .awsConfiguration(AWSMobileClient.getInstance().configuration)
        .build()
  }
}
```

3. Add the project data model files you downloaded from the Mobile Hub console. The data models provide set and get methods for each attribute of a DynamoDB table they model.
   a. Copy the data model file(s) you downloaded, `./YOUR-PROJECT-NAME-integration-lib-aws-my-sample-app-android/src/main/java/com/amazonaws/models/nosqlYOUR-TABLE-NAMEDO.java` into the Android Studio folder that contains your main activity.

Note
Use Asynchronous Calls to DynamoDB

Since calls to DynamoDB are synchronous, they don't belong on your UI thread. Use an asynchronous method like the `thread` wrapper to call `DynamoDBObjectMapper` in a separate thread.

```
thread(start = true) {
    // DynamoDB calls go here
}
```

iOS - Swift

1. Set up AWS Mobile SDK components with the following Set Up Your Backend (p. 2) steps.
   a. `Podfile` that you configure to install the AWS Mobile SDK must contain:

```
platform :ios, '9.0'

target :'YOUR-APP-NAME' do
  use_frameworks!

    pod 'AWSDynamoDB', '~> 2.6.13'
    # other pods
end
```

   Run `pod install --repo-update` before you continue.

   If you encounter an error message that begins "`[!] Failed to connect to GitHub to update the CocoaPods/Specs . . .`", and your internet connectivity is working, you may need to update openssl and Ruby.

   b. Classes that call DynamoDB APIs must use the following import statements:

```
import AWSCore
import AWSDynamoDB
```

2. From the location where you downloaded the data model file(s), drag and drop each file with the form of *`your-table-name`*`.swift` into the folder that contains your `AppDelegate.swift`. Select **Copy items if needed** and **Create groups**, if these options are offered.

Perform CRUD Operations

Topics

Using the Data Model

To connect your app to an Amazon DynamoDB table you have created, use a data model generated by Mobile Hub, or create one in the following form. As an example, the fragments in the following sections are based on a table named `News`. The table's partition key (hash key) is named `userID`, the sort key (range key) is called `articleId` and other attributes, including `author`, `title`, `category`, `content`, and `content`.

Android - Java

In the following example, the `NewsDO` class defines the data model of the `News` table. The class is used by the CRUD methods in this section to access the table and its attributes. The data model file you downloaded from Mobile Hub in previous steps contains a similar class that defines the model of your table.

Note that the class is annotated to map it to the Amazon DynamoDB table name. The attribute names, hash key, and range key of the getters in the class are annotated to map them to local variable names used by the app for performing data operations.

```
package com.amazonaws.models.nosql;

import com.amazonaws.mobileconnectors.dynamodbv2.dynamodbmapper.DynamoDBAttribute;
import com.amazonaws.mobileconnectors.dynamodbv2.dynamodbmapper.DynamoDBHashKey;
import com.amazonaws.mobileconnectors.dynamodbv2.dynamodbmapper.DynamoDBIndexHashKey;
import com.amazonaws.mobileconnectors.dynamodbv2.dynamodbmapper.DynamoDBIndexRangeKey;
import com.amazonaws.mobileconnectors.dynamodbv2.dynamodbmapper.DynamoDBRangeKey;
import com.amazonaws.mobileconnectors.dynamodbv2.dynamodbmapper.DynamoDBTable;

import java.util.List;
import java.util.Map;
import java.util.Set;

@DynamoDBTable(tableName = "nosqlnews-mobilehub-1234567890-News")

public class NewsDO {
    private String _userId;
    private String _articleId;
    private String _author;
    private String _category;
    private String _content;
    private Double _creationDate;
    private String _title;

    @DynamoDBHashKey(attributeName = "userId")
    @DynamoDBAttribute(attributeName = "userId")
    public String getUserId() {
        return _userId;
    }

    public void setUserId(final String _userId) {
        this._userId = _userId;
    }
    @DynamoDBRangeKey(attributeName = "articleId")
    @DynamoDBAttribute(attributeName = "articleId")
    public String getArticleId() {
        return _articleId;
    }

    public void setArticleId(final String _articleId) {
        this._articleId = _articleId;
    }
    @DynamoDBAttribute(attributeName = "author")
    public String getAuthor() {
        return _author;
    }

    public void setAuthor(final String _author) {
        this._author = _author;
    }

    // setters and getters for other attribues ...
```

```
}
```

Android - Kotlin

In the following example, the `NewsDO` class defines the data model of the `News` table. The class is used by the CRUD methods in this section to access the table and its attributes. The data model file you downloaded from Mobile Hub in previous steps contains a similar class that defines the model of your table.

Note that the class is annotated to map it to the Amazon DynamoDB table name. The attribute names, hash key, and range key of the getters in the class are annotated to map them to local variable names used by the app for performing data operations.

```
package com.amazonaws.models.nosql;

import com.amazonaws.mobileconnectors.dynamodbv2.dynamodbmapper.DynamoDBAttribute;
import com.amazonaws.mobileconnectors.dynamodbv2.dynamodbmapper.DynamoDBHashKey;
import com.amazonaws.mobileconnectors.dynamodbv2.dynamodbmapper.DynamoDBIndexHashKey;
import com.amazonaws.mobileconnectors.dynamodbv2.dynamodbmapper.DynamoDBIndexRangeKey;
import com.amazonaws.mobileconnectors.dynamodbv2.dynamodbmapper.DynamoDBRangeKey;
import com.amazonaws.mobileconnectors.dynamodbv2.dynamodbmapper.DynamoDBTable;

import java.util.List;
import java.util.Map;
import java.util.Set;

@DynamoDBTable(tableName = "nosqlnews-mobilehub-1234567890-News")

data class NewsDO {
    @DynamoDBHashKey(attributeName = "userId" )
    @DynamoDBAttribute(attributeName = "userId")
    var userId: String?

    @DynamoDBRangeKey(attributeName = "articleId")
    @DynamoDBAttribute(attributeName = "articleId")
    var articleId: String?

    @DynamoDBAttribute(attributeName = "author")
    var author: String?

    // setters and getters for other attribues ...
}
```

If you download an Android model file generated by Mobile Hub, it will be provided in Java and can be used in a Kotlin project without modifications.

iOS - Swift

In the following example, the `News` class defines the data model of the `News` table. The class is used by the CRUD methods in this section to access the table and its attributes. The data model file you downloaded from Mobile Hub in previous steps contains a similar class that defines the model of your table.

Note that the functions of the model class return the Amazon DynamoDB table, hash key attibute, and range key attribute names used by the app for data operations. For example, `dynamoDBTableName()` returns the name of the table object in AWS. The local variable names map to the attribute names of the table. For instance, `userId` is the name of both the local variable and the attribute of the Amazon DynamoDB table.

This example is slightly simpler than the data model generated by Mobile Hub, but functionally the same.

```
// News.swift

import Foundation
import UIKit
import AWSDynamoDB

class News: AWSDynamoDBObjectModel, AWSDynamoDBModeling {

    @objc var userId: String?
    @objc var articleId: String?
    @objc var author: String?
    @objc var category: String?
    @objc var content: String?
    @objc var creationDate: NSNumber?
    @objc var title: String?

    class func dynamoDBTableName() -> String {

        return "nosqlnews-mobilehub-1200412570-News"
    }

    class func hashKeyAttribute() -> String {

        return "userId"
    }

    class func rangeKeyAttribute() -> String {

        return "articleId"
    }

}
```

Create (Save) an Item

Use the following code to create an item in your NoSQL Database table.

Android - Java

```
public void createNews() {
    final NewsDO newsItem = new NewsDO();

    newsItem.setUserId(unique-user-id);

    newsItem.setArticleId("Article1");
    newsItem.setContent("This is the article content");

    new Thread(new Runnable() {
        @Override
        public void run() {
            dynamoDBMapper.save(newsItem);
                // Item saved
        }
    }).start();
}
```

Android - Kotlin

```
   fun createNews() {
       val NewsDO newsItem = NewsDO()
       newsItem.userId = "unique-user-id"
```

```
        newsItem.articleId = UUID.randomUUID().toString()
        newsItem.author = "Your Name"
        newsItem.content = "This is the article content"

        thread(start = true) {
            dynamoDBMapper.save(newsItem)
        }
    }
```

iOS - Swift

```
func createNews() {
    let dynamoDbObjectMapper = AWSDynamoDBObjectMapper.default()

    // Create data object using data models you downloaded from Mobile Hub
    let newsItem: News = News()

    newsItem.userId = AWSIdentityManager.default().identityId

    newsItem.articleId = "YourArticleId"
    newsItem.title = "YourTitlestring"
    newsItem.author = "YourAuthor"
    newsItem.creationDate = NSDate().timeIntervalSince1970 as NSNumber

    //Save a new item
    dynamoDbObjectMapper.save(newsItem, completionHandler: {
     (error: Error?) -> Void in

         if let error = error {
             print("Amazon DynamoDB Save Error: \(error)")
             return
         }
         print("An item was saved.")
     })
}
```

Read (Load) an Item

Use the following code to read an item in your NoSQL Database table.

Android - Java

```
public void readNews() {
    new Thread(new Runnable() {
        @Override
        public void run() {

            NewsDO newsItem = dynamoDBMapper.load(
                    NewsDO.class,
                    unique-user-id,
                    "Article1");

            // Item read
            // Log.d("News Item:", newsItem.toString());
        }
    }).start();
}
```

Android - Kotlin

```
   fun readNews(userId: String, articleId: String, callback: (NewsDO?) -> Unit) {
```

```
        thread(start = true) {
            var newsItem = dynamoDBMapper.load(NewsDO::class.java,
                    userId, articleId)
            runOnUiThread { callback(newsItem) }
        }
    }
```

iOS - Swift

```
func readNews() {
  let dynamoDbObjectMapper = AWSDynamoDBObjectMapper.default()

      // Create data object using data models you downloaded from Mobile Hub
      let newsItem: News = News();
      newsItem.userId = AWSIdentityManager.default().identityId

      dynamoDbObjectMapper.load(
         News.self,
         hashKey: newsItem.userId,
         rangeKey: "YourArticleId",
         completionHandler: {
            (objectModel: AWSDynamoDBObjectModel?, error: Error?) -> Void in
            if let error = error {
                 print("Amazon DynamoDB Read Error: \(error)")
                 return
             }
             print("An item was read.")
         })
 }
```

Update an Item

Use the following code to update an item in your NoSQL Database table.

Android - Java

```
public void updateNews() {
    final NewsDO newsItem = new NewsDO();

    newsItem.setUserId(unique-user-id);

    newsItem.setArticleId("Article1");
    newsItem.setContent("This is the updated content.");

    new Thread(new Runnable() {
        @Override
        public void run() {

            dynamoDBMapper.save(newsItem);

            // Item updated
        }
    }).start();
}
```

Android - Kotlin

```
    fun updateNews(updatedNews: NewsDO) {
        thread(start = true) {
            dynamoDBMapper.save(updatedNews)
        }
```

```
    }
```

iOS - Swift

```
func updateNews() {
    let dynamoDbObjectMapper = AWSDynamoDBObjectMapper.default()

    let newsItem: News = News();

    newsItem.userId = "unique-user-id"

    newsItem.articleId = "YourArticleId"
    newsItem.title = "This is the Title"
    newsItem.author = "B Smith"
    newsItem.creationDate = NSDate().timeIntervalSince1970 as NSNumber
    newsItem.category = "Local News"

    dynamoDbObjectMapper.save(newsItem, completionHandler: {(error: Error?) -> Void in
        if let error = error {
            print(" Amazon DynamoDB Save Error: \(error)")
            return
        }
        print("An item was updated.")
    })
}
```

Delete an Item

Use the following code to delete an item in your NoSQL Database table.

Android - Java

```
public void deleteNews() {
    new Thread(new Runnable() {
        @Override
        public void run() {

            NewsDO newsItem = new NewsDO();

            newsItem.setUserId(unique-user-id);    //partition key

            newsItem.setArticleId("Article1");  //range (sort) key

            dynamoDBMapper.delete(newsItem);

            // Item deleted
        }
    }).start();
}
```

Android - Kotlin

```
 public void deleteNews(userId: String, articleId: String) {
   thread(start = true) {
       val item = NewsDO()
       item.userId = userId
       item.articleId = articleId

       dynamoDBMapper.delete(item)
   }
 }
```

iOS - Swift

```
func deleteNews() {
    let dynamoDbObjectMapper = AWSDynamoDBObjectMapper.default()

    let itemToDelete = News()
    itemToDelete?.userId = "unique-user-id"
    itemToDelete?.articleId = "YourArticleId"

    dynamoDbObjectMapper.remove(itemToDelete!, completionHandler: {(error: Error?) ->
 Void in
        if let error = error {
            print(" Amazon DynamoDB Save Error: \(error)")
            return
        }
        print("An item was deleted.")
    })
}
```

Perform a Query

A query operation enables you to find items in a table. You must define a query using both the hash key (partition key) and range key (sort key) attributes of a table. You can filter the results by specifying the attributes you are looking for.

The following example code shows querying for news submitted with `userId` (hash key) and article ID beginning with `Trial` (range key).

Android - Java

```
public void queryNews() {

   new Thread(new Runnable() {
       @Override
       public void run() {
           NewsDO news = new NewsDO();
           news.setUserId(unique-user-id);
           news.setArticleId("Article1");

           Condition rangeKeyCondition = new Condition()
                   .withComparisonOperator(ComparisonOperator.BEGINS_WITH)
                   .withAttributeValueList(new AttributeValue().withS("Trial"));

           DynamoDBQueryExpression queryExpression = new DynamoDBQueryExpression()
                   .withHashKeyValues(note)
                   .withRangeKeyCondition("articleId", rangeKeyCondition)
                   .withConsistentRead(false);

           PaginatedList<NewsDO> result = dynamoDBMapper.query(NewsDO.class,
 queryExpression);

           Gson gson = new Gson();
           StringBuilder stringBuilder = new StringBuilder();

           // Loop through query results
           for (int i = 0; i < result.size(); i++) {
               String jsonFormOfItem = gson.toJson(result.get(i));
               stringBuilder.append(jsonFormOfItem + "\n\n");
           }

           // Add your code here to deal with the data result
```

```
            Log.d("Query result: ", stringBuilder.toString());

            if (result.isEmpty()) {
                // There were no items matching your query.
            }
        }
    }).start();
}
```

Android - Kotlin

```
public void queryNews(userId: String, articleId: String, callback: (List<NewsDO>?) ->
 Unit) {
   thread(start = true) {
       val item = NewsDO()
       item.userId = userId
       item.articleId = articleId

       val rangeKeyCondition = Condition()
           .withComparisonOperator(ComparisonOperator.BEGINS_WITH)
           .withAttributeValueList(AttributeValue().withS("Trial"))
       val queryExpression = DynamoDBQueryExpression()
                   .withHashKeyValues(item)
                   .withRangeKeyCondition("articleId", rangeKeyCondition)
                   .withConsistentRead(false);
       val result = dynamoDBMapper.query(NewsDO::class.java, queryExpression)
       runOnUiThread { callback(result) }
   }
}
```

iOS - Swift

```
func queryNote() {
    // 1) Configure the query
    let queryExpression = AWSDynamoDBQueryExpression()
    queryExpression.keyConditionExpression = "#articleId >= :articleId AND #userId
 = :userId"

    queryExpression.expressionAttributeNames = [
         "#userId": "userId",
        "#articleId": "articleId"
    ]
    queryExpression.expressionAttributeValues = [
        ":articleId": "SomeArticleId",
        ":userId": "unique-user-id"
    ]

    // 2) Make the query

    let dynamoDbObjectMapper = AWSDynamoDBObjectMapper.default()

    dynamoDbObjectMapper.query(News.self, expression: queryExpression) { (output:
 AWSDynamoDBPaginatedOutput?, error: Error?) in
      if error != nil {
          print("The request failed. Error: \(String(describing: error))")
      }
      if output != nil {
          for news in output!.items {
              let newsItem = news as? News
              print("\(newsItem!.title!)")
          }
      }
   }
```

```
}
```

Add User File Storage to Your Mobile App with Amazon S3

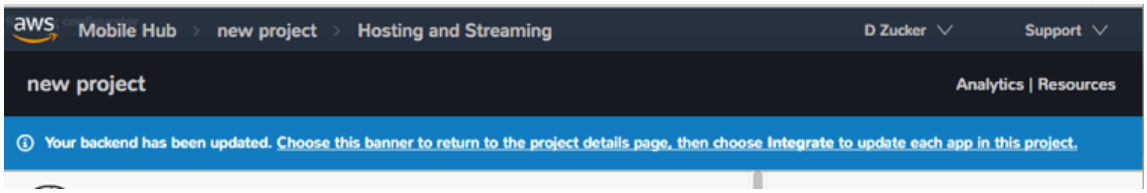
Overview

Enable your app to store and retrieve user files from cloud storage with the permissions model that suits your purpose. Mobile Hub User File Storage (p. 353) deploys and configures cloud storage buckets using Amazon Simple Storage Service (Amazon S3).

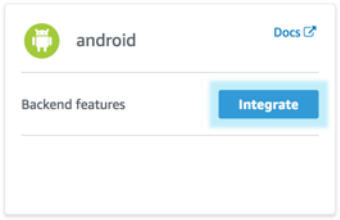
Set Up Your Backend

1. Complete the Get Started (p. 2) steps before your proceed.

   If you want to integrate an Amazon S3 bucket that you have already configured, go to Integrate an Existing Bucket (p. 182).
2. Enable **User File Storage**: Open your project in Mobile Hub and choose the **User File Storage** tile to enable the feature.
3. When the operation is complete, an alert will pop up saying "Your Backend has been updated", prompting you to download the latest copy of the cloud configuration file. If you're done configuring the feature, choose the banner to return to the project details page.



4. From the project detail page, every app that needs to be updated with the latest cloud configuration file will have a flashing **Integrate** button. Choose the button to enter the integrate wizard.



5. Update your app with the latest copy of the cloud configuration file. Your app now references the latest version of your backend. Choose Next and follow the User File Storage documentation below to connect to your backend.

Connect to Your Backend

Make sure to complete the add-aws-mobile-user-sign-in-backend-setup steps before using the integration steps on this page.

To add User File Storage to your app

Android - Java

Set up AWS Mobile SDK components as follows:

1. Add the following to `app/build.gradle` (Module:app):

```
dependencies {
   implementation 'com.amazonaws:aws-android-sdk-s3:2.6.+'
   implementation 'com.amazonaws:aws-android-sdk-cognito:2.6.+'
}
```

   Perform a *Gradle Sync* to download the AWS Mobile SDK components into your app

2. Add the following to `AndroidManifest.xml`:

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />

<application ... >

   <!- Other manifest / application items . . . ->

   <service
 android:name="com.amazonaws.mobileconnectors.s3.transferutility.TransferService"
 android:enabled="true" />

</application>
```

3. For each Activity where you make calls to perform user file storage operations, import the following packages.

```
import com.amazonaws.mobileconnectors.s3.transferutility.*;
```

Android - Kotlin

Set up AWS Mobile SDK components as follows:

1. Add the following to `app/build.gradle`:

```
dependencies {
   implementation 'com.amazonaws:aws-android-sdk-s3:2.6.+'
   implementation 'com.amazonaws:aws-android-sdk-cognito:2.6.+'
}
```

   Perform a *Gradle Sync* to download the AWS Mobile SDK components into your app

2. Add the following to `AndroidManifest.xml`:

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />

<application ... >

   <!- Other manifest / application items . . . ->

   <service
 android:name="com.amazonaws.mobileconnectors.s3.transferutility.TransferService"
 android:enabled="true" />

</application>
```

3. For each Activity where you make calls to perform user file storage operations, import the following packages.

```
import com.amazonaws.mobileconnectors.s3.transferutility.*;
```

iOS - Swift

Set up AWS Mobile SDK components as follows:

1. Add the following to `Podfile` that you configure to install the AWS Mobile SDK:

```
platform :ios, '9.0'

   target :'YOUR-APP-NAME' do
      use_frameworks!

      pod 'AWSS3', '~> 2.6.13'   # For file transfers
      pod 'AWSCognito', '~> 2.6.13'   #For data sync

      # other pods . . .

   end
```

   Run `pod install --repo-update` before you continue.

   If you encounter an error message that begins "`[!] Failed to connect to GitHub to update the CocoaPods/Specs . . .`", and your internet connectivity is working, you may need to update openssl and Ruby.
2. Add the following imports to the classes that perform user file storage operations:

```
import AWSCore
import AWSS3
```

3. Add the following code to your AppDelegate to establish a run-time connection with AWS Mobile.

```
import UIKit
import AWSMobileClient

@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate {

 func application(_ application: UIApplication,
     didFinishLaunchingWithOptions launchOptions: [UIApplicationLaunchOptionsKey:
 Any]?) -> Bool {
         return AWSMobileClient.sharedInstance().interceptApplication(application,
 didFinishLaunchingWithOptions: launchOptions)
 }
}
```

Upload a File

Android - Java

To upload a file to an Amazon S3 bucket, use `AWSMobileClient` to get the `AWSConfiguration` and `AWSCredentialsProvider`, then create the `TransferUtility` object. `AWSMobileClient` expects an activity context for resuming an authenticated session and creating the credentials provider.

The following example shows using the `TransferUtility` in the context of an Activity. If you are creating `TransferUtility` from an application context, you can construct the `AWSCredentialsProvider` and pass it into `TransferUtility` to use in forming the `AWSConfiguration` object. `TransferUtility` will check the size of file being uploaded and will automatically switch over to using multi-part uploads if the file size exceeds 5 MB.

```
import android.app.Activity;
import android.util.Log;

import com.amazonaws.mobile.client.AWSMobileClient;
import com.amazonaws.mobileconnectors.s3.transferutility.TransferUtility;
import com.amazonaws.mobileconnectors.s3.transferutility.TransferState;
import com.amazonaws.mobileconnectors.s3.transferutility.TransferObserver;
import com.amazonaws.mobileconnectors.s3.transferutility.TransferListener;
import com.amazonaws.services.s3.AmazonS3Client;

import java.io.File;

public class YourActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        AWSMobileClient.getInstance().initialize(this).execute();
        uploadWithTransferUtility();
    }

    public void uploadWithTransferUtility() {

        TransferUtility transferUtility =
            TransferUtility.builder()
                .context(getApplicationContext())
                .awsConfiguration(AWSMobileClient.getInstance().getConfiguration())
                .s3Client(new
AmazonS3Client(AWSMobileClient.getInstance().getCredentialsProvider()))
                .build();

        TransferObserver uploadObserver =
            transferUtility.upload(
                "s3Folder/s3Key.txt",
                new File("/path/to/file/localFile.txt"));

        // Attach a listener to the observer to get state update and progress
notifications
        uploadObserver.setTransferListener(new TransferListener() {

            @Override
            public void onStateChanged(int id, TransferState state) {
                if (TransferState.COMPLETED == state) {
                    // Handle a completed upload.
                }
            }

            @Override
            public void onProgressChanged(int id, long bytesCurrent, long bytesTotal)
{
                float percentDonef = ((float) bytesCurrent / (float) bytesTotal) *
100;
                int percentDone = (int)percentDonef;

                Log.d("YourActivity", "ID:" + id + " bytesCurrent: " + bytesCurrent
                        + " bytesTotal: " + bytesTotal + " " + percentDone + "%");
            }

            @Override
            public void onError(int id, Exception ex) {
                // Handle errors
            }

        });

        // If you prefer to poll for the data, instead of attaching a
```

```
          // listener, check for the state and progress in the observer.
          if (TransferState.COMPLETED == uploadObserver.getState()) {
              // Handle a completed upload.
          }

          Log.d("YourActivity", "Bytes Transferrred: " +
 uploadObserver.getBytesTransferred());
          Log.d("YourActivity", "Bytes Total: " + uploadObserver.getBytesTotal());
    }
}
```

Android - Kotlin

To upload a file to an Amazon S3 bucket, use `AWSMobileClient` to get the `AWSConfiguration` and `AWSCredentialsProvider`, then create the `TransferUtility` object. `AWSMobileClient` expects an activity context for resuming an authenticated session and creating the credentials provider.

The following example shows using the `TransferUtility` in the context of an Activity.

If you are creating `TransferUtility` from an application context, you can construct the `AWSCredentialsProvider` and pass it into `TransferUtility` to use in forming the `AWSConfiguration` object. `TransferUtility` will check the size of file being uploaded and will automatically switch over to using multi-part uploads if the file size exceeds 5 MB.

```
import android.app.Activity;
import android.util.Log;

import com.amazonaws.mobile.client.AWSMobileClient;
import com.amazonaws.mobileconnectors.s3.transferutility.TransferUtility;
import com.amazonaws.mobileconnectors.s3.transferutility.TransferState;
import com.amazonaws.mobileconnectors.s3.transferutility.TransferObserver;
import com.amazonaws.mobileconnectors.s3.transferutility.TransferListener;
import com.amazonaws.services.s3.AmazonS3Client;

import java.io.File;

class YourActivity : Activity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        AWSMobileClient.getInstance().initialize(this).execute()
        uploadWithTransferUtility()
    }

    fun uploadWithTransferUtility() {
        val transferUtility = TransferUtility.builder()
            .context(this@YourActivity.applicationContext)
            .awsConfiguration(AWSMobileClient.getInstance().configutation)
            .s3Client(AMazonS3Clinet(AWSMobileClient.getInstance().credentialsProvider)
            .build()

        val uploadObserver = transferUtility.upload("s3folder/s3key.txt", File("/path/
to/localfile.txt"))

        // Attach a listener to the observer
        uploadObserver.transferListener = object : TransferListener() {
            override fun onStateChanged(id: Int, state: TransferState) {
                when(state) {
                    TransferState.COMPLETED -> {
                        /* Handle a completed upload */
                    }
                    else -> {
```

```
                        /* Anything else */
                    }
                }
            }

            override fun onProgressChanged(id: Int, bytesCurrent: Long, bytesTotal:
 Long) {
                val percent = ((bytesCurrent as Float) / (bytesTotal as Float) * 100.0)
 as Int

                Log.d(TAG, "ID: $id is $percent done")
            }

            override fun onError(id: Int, exception: Exception) {
                /* Handle errors */
            }
        }

        // If you prefer to long-poll for updates
        if (uploadObserver.state == TransferState.COMPLETED) {
            /* Handle completion */
        }

        val bytesTransferred = uploadObserver.bytesTransferred
    }
}
```

iOS - Swift

The following example shows how to upload a file to an Amazon S3 bucket.

```
func uploadData() {

   let data: Data = Data() // Data to be uploaded

   let expression = AWSS3TransferUtilityUploadExpression()
      expression.progressBlock = {(task, progress) in
         DispatchQueue.main.async(execute: {
           // Do something e.g. Update a progress bar.
        })
   }

   var completionHandler: AWSS3TransferUtilityUploadCompletionHandlerBlock?
   completionHandler = { (task, error) -> Void in
      DispatchQueue.main.async(execute: {
         // Do something e.g. Alert a user for transfer completion.
         // On failed uploads, `error` contains the error object.
      })
   }

   let transferUtility = AWSS3TransferUtility.default()

   transferUtility.uploadData(data,
        bucket: "YourBucket",
        key: "YourFileName",
        contentType: "text/plain",
        expression: expression,
        completionHandler: completionHandler).continueWith {
           (task) -> AnyObject! in
               if let error = task.error {
                  print("Error: \(error.localizedDescription)")
               }

               if let _ = task.result {
                  // Do something with uploadTask.
```

```
            }
            return nil;
        }
}
```

Download a File

Android - Java

To download a file from an Amazon S3 bucket, use `AWSMobileClient` to get the `AWSConfiguration` and `AWSCredentialsProvider` to create the `TransferUtility` object. `AWSMobileClient` expects an activity context for resuming an authenticated session and creating the `AWSCredentialsProvider`.

The following example shows using the `TransferUtility` in the context of an Activity. If you are creating `TransferUtility` from an application context, you can construct the `AWSCredentialsProvider` and pass it into `TransferUtility` to use in forming the `AWSConfiguration` object.

```
import android.app.Activity;
import android.util.Log;

import com.amazonaws.mobile.client.AWSMobileClient;
import com.amazonaws.mobileconnectors.s3.transferutility.TransferUtility;
import com.amazonaws.mobileconnectors.s3.transferutility.TransferState;
import com.amazonaws.mobileconnectors.s3.transferutility.TransferObserver;
import com.amazonaws.mobileconnectors.s3.transferutility.TransferListener;
import com.amazonaws.services.s3.AmazonS3Client;

import java.io.File;

public class YourActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        AWSMobileClient.getInstance().initialize(this).execute();
        downloadWithTransferUtility();
    }

    private void downloadWithTransferUtility() {

        TransferUtility transferUtility =
            TransferUtility.builder()
                    .context(getApplicationContext())
                    .awsConfiguration(AWSMobileClient.getInstance().getConfiguration())
                    .s3Client(new
 AmazonS3Client(AWSMobileClient.getInstance().getCredentialsProvider()))
                    .build();

        TransferObserver downloadObserver =
            transferUtility.download(
                    "s3Folder/s3Key.txt",
                    new File("/path/to/file/localFile.txt"));

        // Attach a listener to the observer to get state update and progress
 notifications
        downloadObserver.setTransferListener(new TransferListener() {

            @Override
            public void onStateChanged(int id, TransferState state) {
                if (TransferState.COMPLETED == state) {
                    // Handle a completed upload.
```

```
                }
            }

            @Override
            public void onProgressChanged(int id, long bytesCurrent, long bytesTotal) {
                    float percentDonef = ((float)bytesCurrent/(float)bytesTotal) * 100;
                    int percentDone = (int)percentDonef;

                    Log.d("MainActivity", "   ID:" + id + "   bytesCurrent: " +
 bytesCurrent + "   bytesTotal: " + bytesTotal + " " + percentDone + "%");
            }

            @Override
            public void onError(int id, Exception ex) {
                // Handle errors
            }

        });

        // If you prefer to poll for the data, instead of attaching a
        // listener, check for the state and progress in the observer.
        if (TransferState.COMPLETED == downloadObserver.getState()) {
            // Handle a completed upload.
        }

        Log.d("YourActivity", "Bytes Transferrred: " +
 downloadObserver.getBytesTransferred());
        Log.d("YourActivity", "Bytes Total: " + downloadObserver.getBytesTotal());
    }
}
```

Android - Kotlin

To download a file from an Amazon S3 bucket, use `AWSMobileClient` to get the `AWSConfiguration` and `AWSCredentialsProvider` to create the `TransferUtility` object. `AWSMobileClient` expects an activity context for resuming an authenticated session and creating the `AWSCredentialsProvider`.

The following example shows using the `TransferUtility` in the context of an Activity. If you are creating `TransferUtility` from an application context, you can construct the `AWSCredentialsProvider` and pass it into `TransferUtility` to use in forming the `AWSConfiguration` object.

```
import android.app.Activity;
import android.util.Log;

import com.amazonaws.mobile.client.AWSMobileClient;
import com.amazonaws.mobileconnectors.s3.transferutility.TransferUtility;
import com.amazonaws.mobileconnectors.s3.transferutility.TransferState;
import com.amazonaws.mobileconnectors.s3.transferutility.TransferObserver;
import com.amazonaws.mobileconnectors.s3.transferutility.TransferListener;
import com.amazonaws.services.s3.AmazonS3Client;

import java.io.File;

class YourActivity : Activity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_your)

        AWSMobileClient.getInstance().initialize(this).execute()
        donwloadWithTransferUtility()
    }
```

```
    private fun downloadWithTransferUtility() {
        val transferUtility = TransferUtility.builder()
            .context(applicationContext)
            .awsConfiguration(AWSMobileClient.getInstance().configuration)

.s3Client(AmazonS3Client(AWSMobileClient.getInstance().credentialsProvider))
            .build()
        val downloadObserver = transferUtility.download(
            "s3folder/s3key.txt",
            File("/path/to/file/localfile.txt"))

        // Attach a listener to get state updates
        downloadObserver.transferListener = object : TransferListener() {
            override fun onStateChanged(id: Int, state: TransferState) =
                when(state) {
                    TransferState.COMPLETED -> {
                        /* File has finished downloading */
                    }
                    else -> { /* Something else happened */ }
                }

            override fun onProgressChanged(id: Int, bytesCurrent: Long, bytesTotal:
 Long) {
                val percent = ((bytesCurrent as Float) / (bytesTotal as Float) * 100.0)
 as Int

                Log.d(TAG, "ID: $id is $percent done")
            }

            override fun onError(id: Int, exception: Exception) {
                /* Handle errors */
            }
        }

        // If you prefer to poll for the data, instead of attaching a
        // listener, check for the state and progress in the observer.
        if (downloadObserver.state == TransferState.COMPLETED) {
            // Handle a completed upload.
        }

        Log.d("YourActivity", "Bytes Transferrred:
 ${downloadObserver.bytesTransferred}");
    }
}
```

iOS - Swift

The following example shows how to download a file from an Amazon S3 bucket.

```
func downloadData() {
   let expression = AWSS3TransferUtilityDownloadExpression()
   expression.progressBlock = {(task, progress) in DispatchQueue.main.async(execute: {
        // Do something e.g. Update a progress bar.
      })
   }

   var completionHandler: AWSS3TransferUtilityDownloadCompletionHandlerBlock?
   completionHandler = { (task, URL, data, error) -> Void in
      DispatchQueue.main.async(execute: {
        // Do something e.g. Alert a user for transfer completion.
        // On failed downloads, `error` contains the error object.
      })
   }

   let transferUtility = AWSS3TransferUtility.default()
```

```
   transferUtility.downloadData(
         fromBucket: "YourBucket",
         key: "YourFileName",
         expression: expression,
         completionHandler: completionHandler
         ).continueWith {
            (task) -> AnyObject! in if let error = task.error {
              print("Error: \(error.localizedDescription)")
            }

            if let _ = task.result {
              // Do something with downloadTask.

            }
            return nil;
        }
}
```

Next Steps

- For further information about TransferUtility capabilities, see Transfer Files and Data Using TransferUtility and Amazon S3 (p. 194).
- For sample apps that demonstrate TransferUtility capabilities, see Android S3 TransferUtility Sample and iOS S3 TransferUtility Sample.
- Looking for Amazon Cognito Sync? If you are a new user, use AWS AppSync instead. AppSync is a new service for synchronizing application data across devices. Like Cognito Sync, AppSync enables synchronization of a user's own data, such as game state or app preferences. AppSync extends these capabilities by allowing multiple users to synchronize and collaborate in real-time on shared data, such as a virtual meeting space or chatroom. Start building with AWS AppSync now

Add Cloud APIs to Your Mobile App with Amazon API GateWay and AWS Lambda

Cloud Logic Overview

Add RESTful APIs handled by your serverless Lambda functions to extend your mobile app to the range of AWS services and beyond. In Mobile Hub, enabling the Cloud Logic (p. 332) feature uses Amazon API Gateway and AWS Lambda services to provide these capabilities.

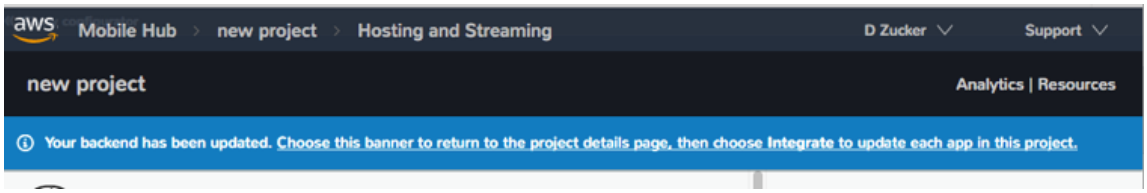
Set Up Your Backend

1. Complete the Get Started (p. 2) steps before your proceed.
2. Enable **Cloud Logic**: Open your project in Mobile Hub and choose the **Cloud Logic** tile to enable the feature.
3. Create a new API or import one that you created in the API Gateway console.
   a. To create a new API choose **Create an API**.
   b. Type an **API Name** and **Description**.
   c. Configure your **Paths**. Paths are locations to the serverless AWS Lambda functions that handle requests to your API.

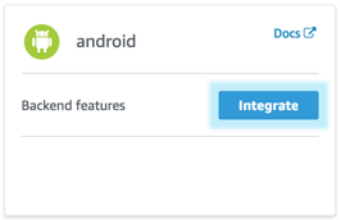
      Choose **Create API** to deploy a default API and its associated handler function. The default handler is a Node.js function that echoes JSON input that it receives. For more information, see Using AWS Lambda with Amazon API Gateway.

The definition of APIs and paths configured in a Mobile Hub project are captured in an AWS CloudFormation template. The body of a request containing a template is limited to 51,200 bytes, see AWS CloudFormation Limits for details. If your API definition is too large to fit this size, you can use the AWS API Gateway Console to create your API and the import it into your Mobile Hub project.

4. When you are done configuring the feature and the last operation is complete, choose your project name in the upper left to go the project details page. The banner that appears also links there.



5. Choose **Integrate** on the app card.



If you have created apps for more than one platform, the **Integrate** button of each that is affected by your project changes will flash, indicating that there is an updated configuration file available for each of those versions.

6. Choose **Download Cloud Config** and replace the old the version of `awsconfiguration.json` with the new download. Your app now references the latest version of your backend.
7. Choose **Swift Models** to download API models that were generated for your app. These files provide access to the request surface for the API Gateway API you just created. Choose **Next** and follow the Cloud API documentation below to connect to your backend.

Connect to Your Backend

Use the following steps to add AWS Cloud Logic to your app.

Android - Java

1. Set up AWS Mobile SDK components with the following Set Up Your Backend (p. 2) steps.
   a. Add the following to your `app/build.gradle`:

```
dependencies{

    // other dependencies . . .

    implementation 'com.amazonaws:aws-android-sdk-apigateway-core:2.6.+'

}
```

   b. For each Activity where you make calls to API Gateway, declare the following imports. Replace the portion of the first declaration, denoted here as `idABCD012345.NAME-OF-YOUR-API-MODEL-CLASS`, with class id and name of the API model that you downloaded from your Mobile Hub project.

You can find these values at the top of the `./src/main/java/com/amazonaws/mobile/api/API-CLASS-ID/TestMobileHubClient.java` file of the download.

```
// This statement imports the model class you download from |AMH|.
import com.amazonaws.mobile.api.idABCD012345.NAME-OF-YOUR-API-MODEL-
CLASSMobileHubClient;

import com.amazonaws.mobile.auth.core.IdentityManager;
import com.amazonaws.mobile.config.AWSConfiguration;
import com.amazonaws.mobileconnectors.apigateway.ApiClientFactory;
import com.amazonaws.mobileconnectors.apigateway.ApiRequest;
import com.amazonaws.mobileconnectors.apigateway.ApiResponse;
import com.amazonaws.util.IOUtils;
import com.amazonaws.util.StringUtils;
import java.io.InputStream;
```

c. The location where you downloaded the API model file(s) contains a folder for each Cloud Logic API you created in your Mobile Hub project. The folders are named for the class ID assigned to the API by API Gateway. For each folder:

   i. In a text editor, open `./src/main/java/com/amazonaws/mobile/api/YOUR-API-CLASS-ID/YOUR-API-CLASS-NAMEMobileHubClient.java`.

   ii. Copy the package name at the top of the file with the form: `com.amazonaws.mobile.api.{api-class-id}`.

   iii. In Android Studio, right-choose `app/java`, and then choose **New > Package**.

   iv. Paste the package name you copied in a previous step and choose **OK**.

   v. Drag and drop the contents of the API class folder into the newly created package. The contents include `YOUR-API-CLASS-NAMEMobileHubClient.java` and the `model` folder.

2. Invoke a Cloud Logic API.

   The following code shows how to invoke a Cloud Logic API using your API's client class, model, and resource paths.

```
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
import com.amazonaws.http.HttpMethodName;
import java.io.InputStream;
import java.util.HashMap;

import com.amazonaws.mobile.client.AWSMobileClient;
import com.amazonaws.mobileconnectors.api.YOUR-API-CLASS-ID.YOUR-API-CLASS-
NAMEMobilehubClient;
import com.amazonaws.mobileconnectors.apigateway.ApiClientFactory;
import com.amazonaws.mobileconnectors.apigateway.ApiRequest;
import com.amazonaws.mobileconnectors.apigateway.ApiResponse;
import com.amazonaws.util.StringUtils;


public class MainActivity extends AppCompatActivity {
    private static final String LOG_TAG = MainActivity.class.getSimpleName();

    private YOUR-API-CLASS-NAMEMobileHubClient apiClient;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

         // Create the client
```

```
        apiClient = new ApiClientFactory()

.credentialsProvider(AWSMobileClient.getInstance().getCredentialsProvider())
                        .build(YOUR-API-CLASS-NAMEMobileHubClient.class);
    }


   public callCloudLogic() {
       // Create components of api request
       final String method = "GET";

       final String path = "/items";

       final String body = "";
       final byte[] content = body.getBytes(StringUtils.UTF8);

       final Map parameters = new HashMap<>();
       parameters.put("lang", "en_US");

       final Map headers = new HashMap<>();

       // Use components to create the api request
       ApiRequest localRequest =
               new ApiRequest(apiClient.getClass().getSimpleName())
                       .withPath(path)
                       .withHttpMethod(HttpMethodName.valueOf(method))
                       .withHeaders(headers)
                       .addHeader("Content-Type", "application/json")
                       .withParameters(parameters);

       // Only set body if it has content.
       if (body.length() > 0) {
           localRequest = localRequest
                   .addHeader("Content-Length", String.valueOf(content.length))
                   .withBody(content);
       }

       final ApiRequest request = localRequest;

       // Make network call on background thread
       new Thread(new Runnable() {
           @Override
           public void run() {
               try {
                   Log.d(LOG_TAG,
                   "Invoking API w/ Request : " +
                   request.getHttpMethod() + ":" +
                   request.getPath());

                   final ApiResponse response = apiClient.execute(request);

                   final InputStream responseContentStream = response.getContent();

                   if (responseContentStream != null) {
                       final String responseData =
IOUtils.toString(responseContentStream);
                       Log.d(LOG_TAG, "Response : " + responseData);
                   }

                   Log.d(LOG_TAG, response.getStatusCode() + " " +
response.getStatusText());

               } catch (final Exception exception) {
                   Log.e(LOG_TAG, exception.getMessage(), exception);
                   exception.printStackTrace();
               }
```

```
            }
        }).start();
    }
}
```

Android - Kotlin

1. Set up AWS Mobile SDK components with the following Set Up Your Backend (p. 2) steps.
   a. Add the following to your `app/build.gradle`:

```
dependencies{

    // other dependencies . . .

    implementation 'com.amazonaws:aws-android-sdk-apigateway-core:2.6.+'

}
```

   b. For each Activity where you make calls to API Gateway, declare the following imports. Replace the portion of the first declaration, denoted here as `idABCD012345.NAME-OF-YOUR-API-MODEL-CLASS`, with class id and name of the API model that you downloaded from your Mobile Hub project.

      You can find these values at the top of the `./src/main/java/com/amazonaws/mobile/api/API-CLASS-ID/TestMobileHubClient.java` file of the download.

```
// This statement imports the model class you download from |AMH|.
import com.amazonaws.mobile.api.idABCD012345.NAME-OF-YOUR-API-MODEL-
CLASSMobileHubClient;

import com.amazonaws.mobile.auth.core.IdentityManager;
import com.amazonaws.mobile.config.AWSConfiguration;
import com.amazonaws.mobileconnectors.apigateway.ApiClientFactory;
import com.amazonaws.mobileconnectors.apigateway.ApiRequest;
import com.amazonaws.mobileconnectors.apigateway.ApiResponse;
import com.amazonaws.util.IOUtils;
import com.amazonaws.util.StringUtils;
import java.io.InputStream;
```

   c. The location where you downloaded the API model file(s) contains a folder for each Cloud Logic API you created in your Mobile Hub project. The folders are named for the class ID assigned to the API by API Gateway. For each folder:
      i. In a text editor, open `./src/main/java/com/amazonaws/mobile/api/YOUR-API-CLASS-ID/YOUR-API-CLASS-NAMEMobileHubClient.java`.
      ii. Copy the package name at the top of the file with the form: `com.amazonaws.mobile.api.{api-class-id}`.
      iii. In Android Studio, right-choose `app/java`, and then choose **New > Package**.
      iv. Paste the package name you copied in a previous step and choose **OK**.
      v. Drag and drop the contents of the API class folder into the newly created package. The contents include `YOUR-API-CLASS-NAMEMobileHubClient.java` and the `model` folder.
2. Invoke a Cloud Logic API.

   The following code shows how to invoke a Cloud Logic API using your API's client class, model, and resource paths.

```
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
```

```
import android.util.Log;
import com.amazonaws.http.HttpMethodName;
import java.io.InputStream;
import java.util.HashMap;

import com.amazonaws.mobile.client.AWSMobileClient;
import com.amazonaws.mobileconnectors.api.YOUR-API-CLASS-ID.YOUR-API-CLASS-
NAMEMobilehubClient;
import com.amazonaws.mobileconnectors.apigateway.ApiClientFactory;
import com.amazonaws.mobileconnectors.apigateway.ApiRequest;
import com.amazonaws.mobileconnectors.apigateway.ApiResponse;
import com.amazonaws.util.StringUtils;

class MainActivity : AppCompatActivity() {
    companion object {
        private val TAG = this::class.java.simpleName
    }

    private var apiClient: YOUR-API-CLASS-NAMEMobileHubClient? = null

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        apiClient = ApiClientFactory()
            .credentialsProvider(AWSMobileClient.getInstance().credentialsProvider)
            .build(YOUR-API-CLASS-NAMEMobileHubClinet::class.java)
    }

    fun callCloudLogic(body: String) {
        val parameters = mapOf("lang" to "en_US")
        val headers = mapOf("Content-Type" to "application/json")

        val request = ApiRequest(apiClient::class.java.simpleName)
            .withPath("/items")
            .withHttpMethod(HttpMethod.GET)
            .withHeaders(headers)
            .withParameters(parameters)
        if (body.isNotEmpty()) {
            val content = body.getBytes(StringUtils.UTF8)
            request
                .addHeader("Content-Length", String.valueOf(content.length))
                .withBody(content)
        }

        thread(start = true) {
            try {
                Log.d(TAG, "Invoking API")
                val response = apiClient.execute(request)
                val responseContentStream = response.getContent()
                if (responseContentStream != null) {
                    val responseData = IOUtils.toString(responseContentStream)
                    // Do something with the response data here
                }
            } catch (ex: Exception) {
                Log.e(TAG, "Error invoking API")
            }
        }
    }
}
```

iOS - Swift

1. Set up AWS Mobile SDK components with the following Set Up Your Backend (p. 2) steps.

a. `Podfile` that you configure to install the AWS Mobile SDK must contain:

```
platform :ios, '9.0'

target :'YOUR-APP-NAME' do
   use_frameworks!

      pod 'AWSAuthCore', '~> 2.6.13'
      pod 'AWSAPIGateway', '~> 2.6.13'
      # other pods

end
```

Run `pod install --repo-update` before you continue.

If you encounter an error message that begins "`[!] Failed to connect to GitHub to update the CocoaPods/Specs . . .`", and your internet connectivity is working, you may need to update openssl and Ruby.

b. Classes that call API Gateway APIs must use the following import statements:

```
import AWSAuthCore
import AWSCore
import AWSAPIGateway
```

c. Add the backend service configuration and API model files that you downloaded from the Mobile Hub console, The API model files provide an API calling surface for each API Gateway API they model.

i. From the location where you downloaded the data model file(s), drag and drop the `./AmazonAws/API` folder into the Xcode project folder that contains `AppDelegate.swift`.

Select **Copy items if needed** and **Create groups**, if these options are offered.

If your Xcode project already contains a `Bridging_Header.h` file then open `./AmazonAws/Bridging_Header.h`, copy the import statement it contains, and paste it into your version of the file.

If your Xcode project does not contain a `Bridging_Header.h` file then:

A. Drag and drop `./AmazonAws/Bridging_Header.h` into the Xcode project folder that contains `AppDelegate.swift`.

B. Choose your project root in Xcode, then choose **Build Settings**, and search for "bridging headers"

C. Choose **Objective-C Bridging Header**, press your *return* key, and type the path within your Xcode project:

your-project-name/.../`Bridging_Header.h`

2. Invoke a Cloud Logic API.

To invoke a Cloud Logic API, create code in the following form and substitute your API's client class, model, and resource paths.

```
import UIKit
import AWSAuthCore
import AWSCore
import AWSAPIGateway
import AWSMobileClient

// ViewController or application context . . .
```

```
 func doInvokeAPI() {
      // change the method name, or path or the query string parameters here as
desired
      let httpMethodName = "POST"
      // change to any valid path you configured in the API
      let URLString = "/items"
      let queryStringParameters = ["key1":"{value1}"]
      let headerParameters = [
          "Content-Type": "application/json",
          "Accept": "application/json"
      ]

      let httpBody = "{ \n  " +
              "\"key1\":\"value1\", \n  " +
              "\"key2\":\"value2\", \n  " +
              "\"key3\":\"value3\"\n}"

      // Construct the request object
      let apiRequest = AWSAPIGatewayRequest(httpMethod: httpMethodName,
              urlString: URLString,
              queryParameters: queryStringParameters,
              headerParameters: headerParameters,
              httpBody: httpBody)

      // Create a service configuration object for the region your AWS API was
created in
      let serviceConfiguration = AWSServiceConfiguration(
          region: AWSRegionType.USEast1,
          credentialsProvider:
AWSMobileClient.sharedInstance().getCredentialsProvider())

          YOUR-API-CLASS-NAMEMobileHubClient.register(with: serviceConfiguration!,
forKey: "CloudLogicAPIKey")

          // Fetch the Cloud Logic client to be used for invocation
          let invocationClient =
              YOUR-API-CLASS-NAMEMobileHubClient(forKey: "CloudLogicAPIKey")

          invocationClient.invoke(apiRequest).continueWith { (
              task: AWSTask) -> Any? in

              if let error = task.error {
                  print("Error occurred: \(error)")
                  // Handle error here
                  return nil
              }

              // Handle successful result here
              let result = task.result!
              let responseString =
                  String(data: result.responseData!, encoding: .utf8)

              print(responseString)
              print(result.statusCode)

              return nil
          }
      }
```

Add Messaging to Your Mobile App with Amazon Pinpoint

Overview

Engage your users more deeply by tying their app usage behavior to messaging campaigns.

When you enable the AWS Mobile Hub Messaging and Analytics (p. 340) feature, your app is registered with the Amazon Pinpoint service. You can define User Segments and send E-mail, SMS, and Push Notification (p. 43) messages to those recipients through the Amazon Pinpoint console.

Amazon Pinpoint also enables you to gather and visualize your app's Analytics. The metrics you gather can be as simple as session start and stop data, or you can customize them to show things like how closely actual behavior matches your predicted model.

You can then algorithmically tie messaging campaigns to user behavior. For instance, send a discount mail to frequent users, or send a push notification that initiates a data sync for users that have selected a certain category in a feature of your app.

Set Up Your Backend

To set up email or SMS as part of a Amazon Pinpoint campaign take the following steps.

To setup your app to receive Push Notifications from Amazon Pinpoint, see Add Push Notifications to Your Mobile App with Amazon Pinpoint (p. 43)

1. Complete the Get Started (p. 2) steps before your proceed.
2. **For Email: Choose the Messaging and Analytics tile to enable the**

   feature.
   a. Choose **Email**, and then choose **Enable**.
   b. Choose the **Amazon Pinpoint console** link at the bottom of the descriptive text on the left.
   c. Choose **Email** in the Amazon Pinpoint console **Channels** tab.
   d. Choose **Email address**, type the address your messages should come from, and then choose **verify** at the end of the entry field.

      The email account you enter will receive an email requesting your approval for Amazon Pinpoint to use that account as the sender address for emails sent by the system. The status of **Pending Verification** is displayed in the console entry field until Amazon Pinpoint has processed your approval.
   e. Choose **Email domain**, type the domain your messages should come from, and then choose **verify** at the end of the entry field.

      A dialog is displayed providing the name and value of the TXT record you must add to the domain's settings. The status of `Pending Verification` is displayed in the entry field until the console processes your approval.

      Add a default user name to **Default from address**.
   f. Choose **Save**.
   g. For information about sending mail from Amazon Pinpoint, see Sending an Email Message.

a. **For SMS**: Choose the **Messaging and Analytics** tile to enable the feature.
   i. Choose **SMS**, and then choose **Enable**.
   ii. Choose the the **Amazon Pinpoint console** link at the bottom of the descriptive text on the left.
   iii. Choose **SMS** in the Amazon Pinpoint console **Channels** tab.

iv. Adjust the options for **Default message type**, **Account spend limit**, and **Default sender ID**. For more information on these options, see Updating SMS Settings.

v. For information about sending SMS messages from Amazon Pinpoint, see Sending an SMS Message.

Connect to your backend

The AWS Mobile SDK is not required to receive Email or SMS messages from Amazon Pinpoint.

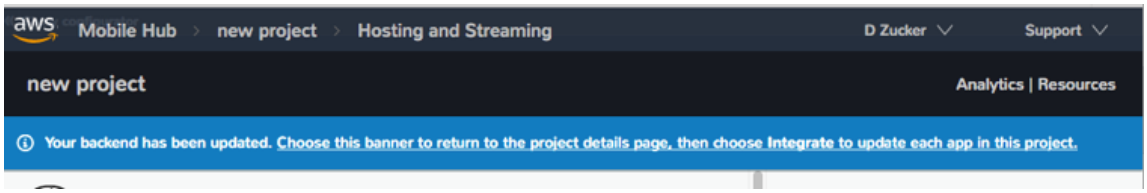
Add Conversational Bots to Your Mobile App with Amazon Lex

Overview

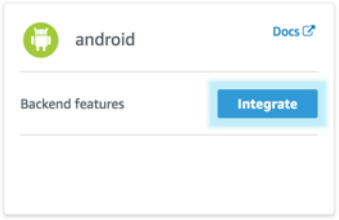
Add the natural language understanding that powers Amazon Alexa to your mobile app. The Mobile HubConversational Bots (p. 346) feature provides ready-made bot templates using the Amazon Lex service.

Set Up Your Backend

1. Complete the Get Started (p. 2) steps before your proceed.
2. Enable **Conversational Bots**: Open your project in Mobile Hub and choose the **Conversational Bots** tile to enable the feature.
   a. Choose one of the sample Bots or import one that you have created in the Amazon Lex console.
3. When the operation is complete, an alert will pop up saying "Your Backend has been updated", prompting you to download the latest copy of the cloud configuration file. If you're done configuring the feature, choose the banner to return to the project details page.



4. From the project detail page, every app that needs to be updated with the latest cloud configuration file will have a flashing **Integrate** button. Choose the button to enter the integrate wizard.



5. Update your app with the latest copy of the cloud configuration file. Your app now references the latest version of your backend. Choose Next and follow the Cloud API documentation below to connect to your backend.

Connect to your backend

To add AWS Mobile Conversational Bots to your app

Android - Java

1. Set up AWS Mobile SDK components with the following Set Up Your Backend (p. 2) steps.
   a. Add the following permissions to your `AndroidManifest.xml`:

```
<uses-permission android:name="android.permission.RECORD_AUDIO" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

   b. Add the following to your `app/build.gradle`:

```
dependencies{
    implementation ('com.amazonaws:aws-android-sdk-lex:2.6.+@aar') {transitive =
 true;}
}
```

   c. For each Activity where you make calls to Amazon Lex, import the following APIs.

```
import com.amazonaws.mobileconnectors.lex.interactionkit.Response;
import com.amazonaws.mobileconnectors.lex.interactionkit.config.InteractionConfig;
import com.amazonaws.mobileconnectors.lex.interactionkit.ui.InteractiveVoiceView;
```

2. Add a voice button to an activity or fragment layout
   a. Add a `voice_component` to your layout file.

```
<com.amazonaws.mobileconnectors.lex.interactionkit.ui.InteractiveVoiceView
android:id="@+id/voiceInterface"
layout="@layout/voice_component"
android:layout_width="200dp"
android:layout_height="200dp"/>
```

   b. In your `strings.xml` file add the region for your bot. *Note: Currently bots are only supported in US Virginia East (us-east-1).*

```
<string name="aws_region">us-east-1</string>
```

   c. Initialize the voice button

      Add the following `init()` function to the `onCreate()` of the activity where your Bot will be used.

      Initialize `AWSMobileClient` before the call to `init()`, as the `InteractiveVoiceView` in the function connects to Amazon Lex using the credentials provider object created by `AWSMobileClient` .

```
import com.amazonaws.mobile.client.AWSMobileClient;
import com.amazonaws.mobile.client.AWSStartupHandler;
import com.amazonaws.mobile.client.AWSStartupResult;
import com.amazonaws.mobileconnectors.lex.interactionkit.Response;
import com.amazonaws.mobileconnectors.lex.interactionkit.config.InteractionConfig;
import com.amazonaws.mobileconnectors.lex.interactionkit.ui.InteractiveVoiceView;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        AWSMobileClient.getInstance().initialize(this, new AWSStartupHandler() {
            @Override
```

```
            public void onComplete(AWSStartupResult awsStartupResult) {
                Log.d("YourMainActivity", "AWSMobileClient is instantiated and you
 are connected to AWS!");
            }
        }).execute();

        init();

    }

    public void init(){
        InteractiveVoiceView voiceView =
                (InteractiveVoiceView) findViewById(R.id.voiceInterface);

        voiceView.setInteractiveVoiceListener(
                new InteractiveVoiceView.InteractiveVoiceListener() {

                    @Override
                    public void dialogReadyForFulfillment(Map slots, String intent)
 {
                        Log.d(LOG_TAG, String.format(
                                Locale.US,
                                "Dialog ready for fulfillment:\n\tIntent: %s\n
\tSlots: %s",
                                intent,
                                slots.toString()));
                    }

                    @Override
                    public void onResponse(Response response) {
                        Log.d(LOG_TAG, "Bot response: " +
 response.getTextResponse());
                    }

                    @Override
                    public void onError(String responseText, Exception e) {
                        Log.e(LOG_TAG, "Error: " + responseText, e);
                    }
                });


 voiceView.getViewAdapter().setCredentialProvider(AWSMobileClient.getInstance().getCredentialsI

        //replace parameters with your botname, bot-alias
        voiceView.getViewAdapter()
                .setInteractionConfig(
                        new InteractionConfig("YOUR-BOT-NAME","$LATEST"));

        voiceView.getViewAdapter()
                .setAwsRegion(getApplicationContext()
                        .getString(R.string.aws_region));
    }
}
```

Android - Kotlin

1. Set up AWS Mobile SDK components with the following Set Up Your Backend (p. 2) steps.
   a. Add the following permissions to your `AndroidManifest.xml`:

```
<uses-permission android:name="android.permission.RECORD_AUDIO" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

   b. Add the following to your `app/build.gradle`:

```
dependencies{
    implementation ('com.amazonaws:aws-android-sdk-lex:2.6.+@aar') {transitive =
true;}
}
```

c. For each Activity where you make calls to Amazon Lex, import the following APIs.

```
import com.amazonaws.mobileconnectors.lex.interactionkit.Response;
import com.amazonaws.mobileconnectors.lex.interactionkit.config.InteractionConfig;
import com.amazonaws.mobileconnectors.lex.interactionkit.ui.InteractiveVoiceView;
```

2. Add a voice button to an activity or fragment layout

a. Add a `voice_component` to your layout file.

```
<com.amazonaws.mobileconnectors.lex.interactionkit.ui.InteractiveVoiceView
android:id="@+id/voiceInterface"
layout="@layout/voice_component"
android:layout_width="200dp"
android:layout_height="200dp"/>
```

b. In your `strings.xml` file add the region for your bot. *Note: Currently bots are only supported in US Virginia East (us-east-1).*

```
<string name="aws_region">us-east-1</string>
```

c. Initialize the voice button

In the `onCreate()` of the activity where your Bot will be used, call `init()`.

```
fun init() {
    voiceInterface.interactiveVoiceListener =
        object : InteractiveVoiceView.InteractiveVoiceListener() {
            override fun dialogReadyFOrFulfillment(slots: Map, intent: String) {
                Log.d(TAG, "Dialog ready for fulfillment:\n\tIntent: $intent")
            }

            override fun onResponse(response: Response) {
                Log.d(TAG, "Bot response: ${response.textResponse}")
            }

            override fun onError(responseText: String, e: Exception) {
                Log.e(TAG, "Error: ${e.message}")
            }
        }

    with (voiceInterface.viewAdapter) {
        credentialsProvider = AWSMobileClient.getInstance().credentialsProvider
        interactionConfig = InteractionConfig("YOUR-BOT-NAME","$LATEST")
        awsRegion = applicationContext.getString(R.string.aws_region)
    }
}
```

iOS - Swift

1. Set up AWS Mobile SDK components with the following Set Up Your Backend (p. 2) steps.

a. `Podfile` that you configure to install the AWS Mobile SDK must contain:

```
platform :ios, '9.0'
```

```
 target :'YOUR-APP-NAME`' do
   use_frameworks!

       pod 'AWSLex', '~> 2.6.13'
       # other pods

end
```

Run `pod install --repo-update` before you continue.

If you encounter an error message that begins "`[!] Failed to connect to GitHub to update the CocoaPods/Specs . . .`", and your internet connectivity is working, you may need to update openssl and Ruby.

b. Classes that call Amazon Lex APIs must use the following import statements:

```
import AWSCore
import AWSLex
```

2. Add permissions to your `info.plist` that allow the app to use the microphone of a device.

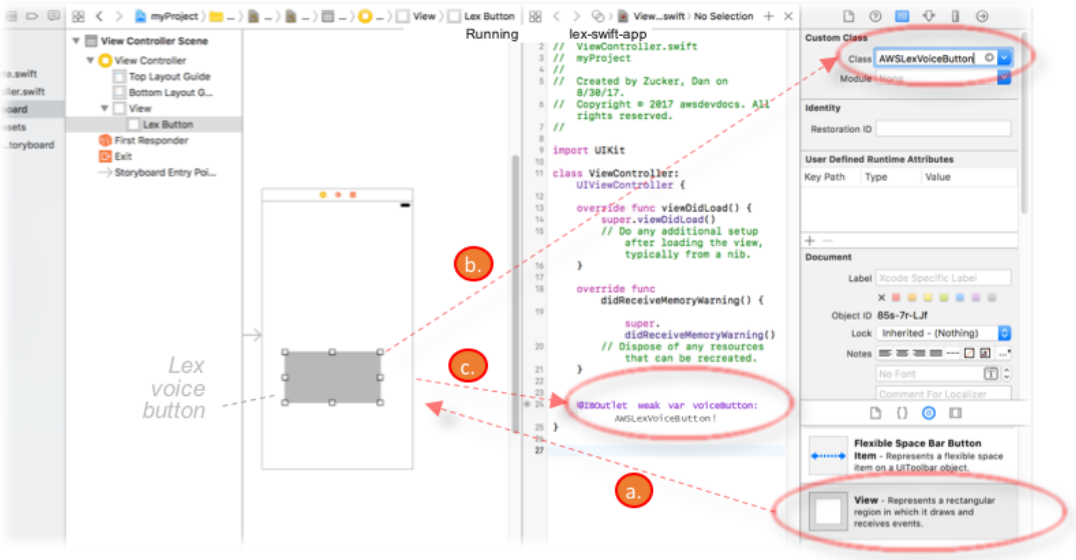
```
<plist version = "1.0"></plist>
   <dict>
      <!-- . . . -->
      <key>NSMicrophoneUsageDescription</key>
      <string>For demonstration of conversational bots</string>
      <!-- . . . -->
   </dict>
```

3. Add your backend service configuration to the app.

   From the location where your Mobile Hub configuration file was downloaded in a previous step, drag `awsconfiguration.json` into the folder containing your `info.plist` file in your Xcode project.

   Select **Copy items if needed** and **Create groups**, if these options are offered.

4. Add a voice button UI element that will let your users speak to Amazon Lex to an activity.
   a. Create a `UIView` in a storyboard or `xib` file.
   b. Map the `UIView` to the `AWSLexVoiceButton` class of the AWS Mobile SDK.
   c. Link the `UIView` to your `ViewController`.



5. Register the voice button.

   The following code shows how use the `viewDidLoad` method of your View Controller to enable your voice button to respond to Amazon Lex success and error messages The code conforms the class to `AWSLexVoiceButtonDelegate`. It initializes the button by binding it to the bot you configured in your Mobile Hub project, and registers the button as the `AWSLexVoiceButtonKey` of your Amazon Lex voice interaction client.

```
import UIKit
import AWSLex
import AWSAuthCore

class VoiceChatViewController: UIViewController, AWSLexVoiceButtonDelegate {
  override func viewDidLoad() {

        // Set the bot configuration details
        // You can use the configuration constants defined in AWSConfiguration.swift
 file
        let botName = "YOUR-BOT-NAME"
        let botRegion: AWSRegionType = "YOUR-BOT-REGION"
        let botAlias = "$LATEST"

        // set up the configuration for AWS Voice Button
        let configuration = AWSServiceConfiguration(region: botRegion,
 credentialsProvider: AWSMobileClient.sharedInstance().getCredentialsProvider())
        let botConfig =
 AWSLexInteractionKitConfig.defaultInteractionKitConfig(withBotName: YOUR-BOT-NAME,
 botAlias: :YOUR-BOT-ALIAS)

        // register the interaction kit client for the voice button using the
 AWSLexVoiceButtonKey constant defined in SDK
        AWSLexInteractionKit.register(with: configuration!,
 interactionKitConfiguration: botConfig, forKey: AWSLexVoiceButtonKey)
        super.viewDidLoad()
        (self.voiceButton as AWSLexVoiceButton).delegate = self
    }
}
```

6. Handle Amazon Lex success and error messages by adding the following delegate methods for the Voice Button in your View Controller.

```
func voiceButton(_ button: AWSLexVoiceButton, on response: AWSLexVoiceButtonResponse)
 {
    // handle response from the voice button here
    print("on text output \(response.outputText)")
}

func voiceButton(_ button: AWSLexVoiceButton, onError error: Error) {
    // handle error response from the voice button here
    print("error \(error)")
}
```

Tutorials

Notes App Tutorial

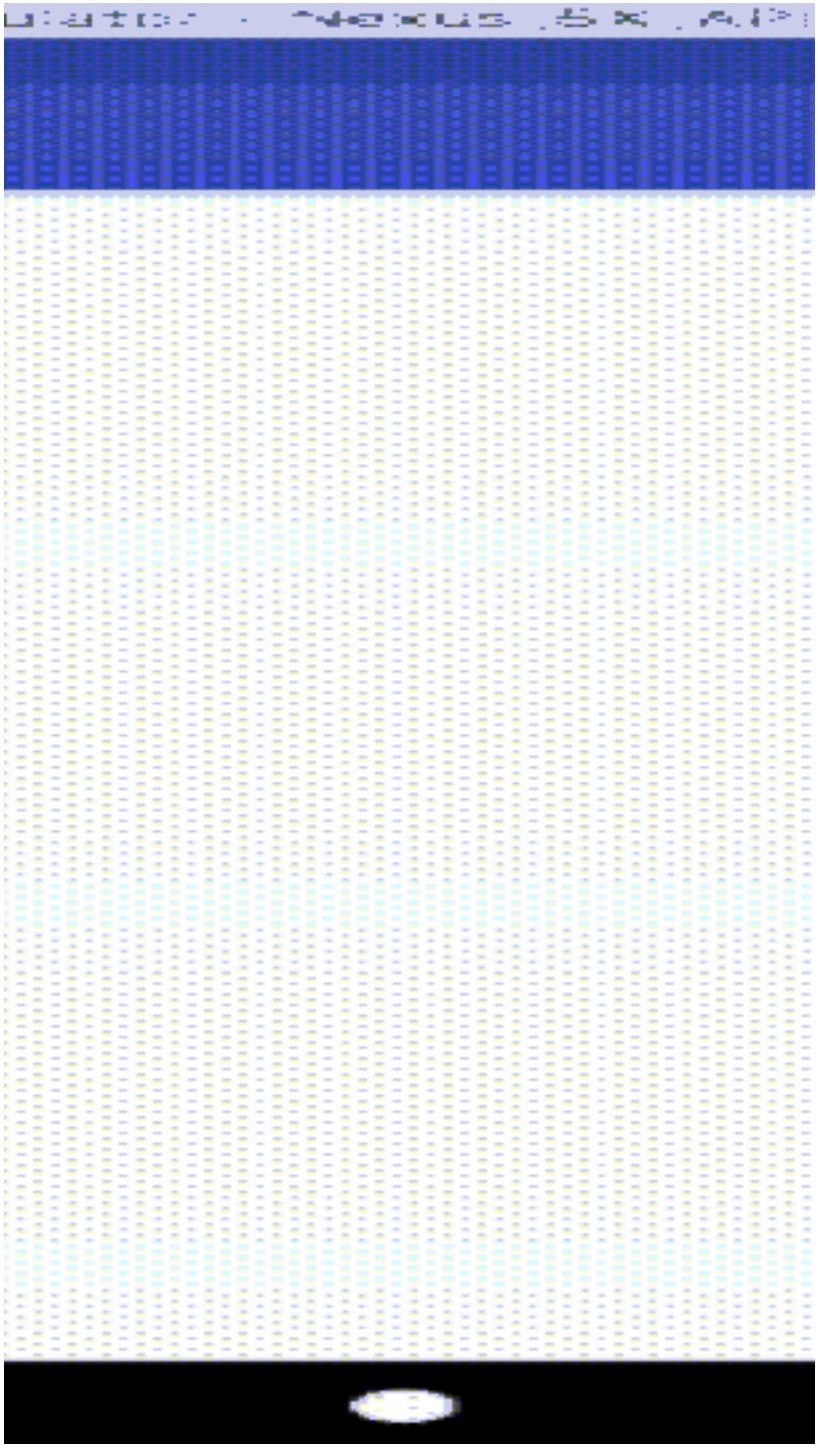
Build a simple note taking application. Start with an offline version of our data-driven app. Cloud-enable the app, adding app analytics, authentication and cloud data storage through AWS services.

- Android Notes Tutorial (p. 90)
- iOS Notes Tutorial (p. 114)

A Simple Note-taking App

Start with a working app and then add cloud enabled features. In this tutorial you will take a working app, driven from locally stored data, and then:

- Add analytics to your app (p. 94) so you can view demographic information about your users.
- Add a simple sign-in/sign-up flow (p. 101) for authentication.
- Access data stores in the AWS (p. 105) cloud, so that a user's notes are available to them on any device with the app installed.



You should be able to complete the setup section of this tutorial within 10-15 minutes after you have installed all required software. Once you complete the instructions on this page, you can run the project on your local system.

Getting Started

Before beginning, you must:

- Download and install Android Studio version 3.0.1 or later.
- Download an install Android SDK version 8.0 (Oreo), API level 26.
- Install an Android Emulator - the app works with both phone and tablet emulators (for example, the Nexus 5X or Pixel C).

Windows Specific Instructions

- Install Git for Windows.

Mac Specific Instructions

- Install XCode using the Mac App Store.
- Configure the XCode command line tools. Run `xcode-select --install` from a Terminal window.

> Why do I need XCode?
>
> The XCode package includes command line tools that are used on a Mac to assist with software development. You don't need to run the UI XCode application.

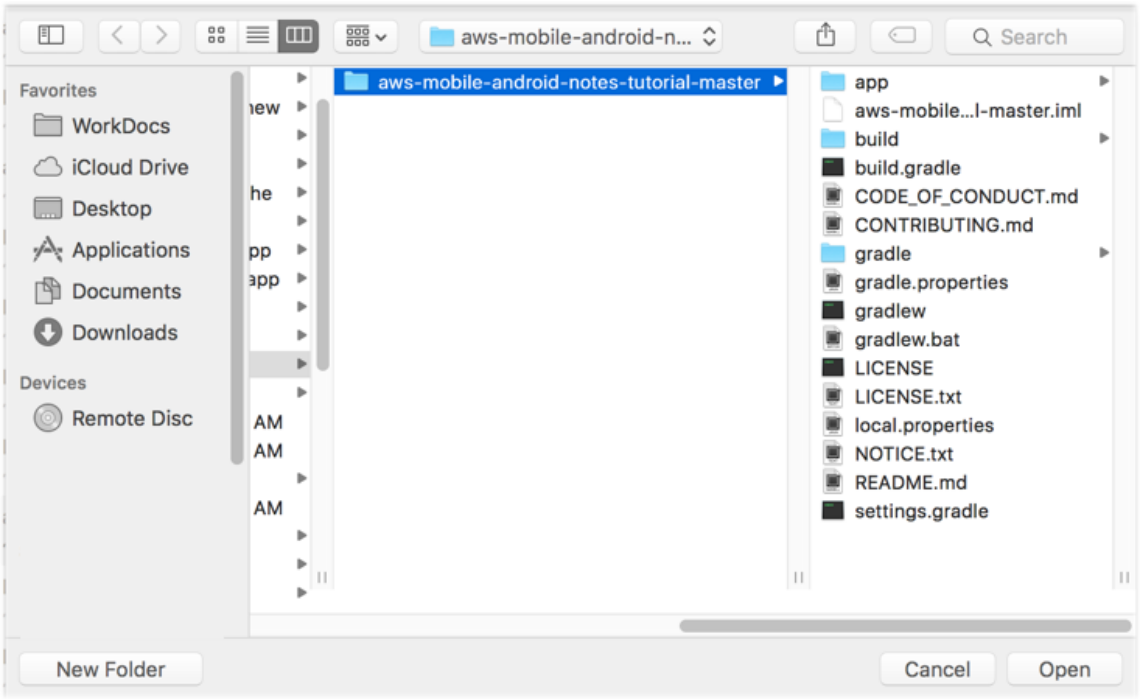
Download the Source code

1. Get the tutorial source code using one of the following choices:
   - Download the source code as a ZIP file.
   - Browse to https://github.com/aws-samples/aws-mobile-android-notes-tutorial/ and clone or fork the repository (sign up for GitHub account, if you do not have one).

Compile the Source Code

To compile the source code:

1. Start **Android Studio**.
2. If you have a project open already, choose **File > Close Project**.
3. Choose **Open an existing Android Studio project**.
4. Find and choose the **aws-mobile-android-notes-tutorial-master** project in your file system, then choose **OK**.



5. Select **Build > Make Project** from the menu bar.

The compilation step should complete with no errors. Errors are displayed within the **Messages** window, available on the status bar at the bottom of the project.

Run the Project in an Emulator

Create a new emulator if you have not done so already:

1. Select **Tools > Android > AVD Manager**.
2. Choose **Create Virtual Device...**.
3. Select **Phone > Nexus 5X**, then choose **Next**.
4. Select the **x86 Images** tab, then select **Android 8.0 (Google APIs)**.
   - If you have not previously downloaded the image, you can download it from this screen.
5. Choose **Next**.
6. Choose **Finish**.
7. Close the AVD Manager.

Run the project in an emulator.

1. Select **Run** > **Run 'app'**.
2. Select the **Nexus 5X API 26** virtual device.
3. Choose **OK**.

The Android emulator will boot (if it is not already started) and the application will run. You should be able to interact with the application as you would any other mobile app. Try pressing on the **+** at the bottom to create a note, or choose a note to show the editor screen. A unique ID for each note is displayed in the list view underneath the note's title.

Running into Problems

The most common problems at this stage involve issues with the installation of Java, Android Studio, the Android SDK or the Android Emulator. Google provides detailed instructions on Android Studio and dependent features.

Next Steps

Next, integrate application analytics (p. 94) into your project.

Add Analytics to the Notes App

In the previous section (p. 90) of this tutorial, we installed Android Studio, downloaded a sample note-taking app from GitHub, then compiled and ran it in the Android Emulator. This tutorial assumes you have completed the those steps. In this section, we will extend the notes app to include application analytics. Application analytics allow us to gather demographic information about the application usage.

You should be able to complete this section in 10-15 minutes.

Set Up Your Back End

To start, set up the mobile backend resources in AWS:

1. Open the AWS Mobile Hub console.
   - If you do not have an AWS account, sign up for the AWS Free Tier.
2. Choose **Create** on the upper left, and the type `android-notes-app` for the name of the Mobile Hub project.
3. Choose **Next**, choose **Android**, and then choose **Add**.
4. Choose **Download Cloud Config**, and save `awsconfiguration.json`. This file the configuration to connect your app to your backend.
5. Choose **Next** and then choose **Done** to create the project.

Used in this section	AWS Mobile Hub: Configure your mobile app's AWS backend in minutes, and then to manage those resources as your app evolves.

Add Permissions to the AndroidManifest.xml

1. Open the project in Android Studio.
2. Choose **Project** on the left side of the project to open the project browser. Find the app manifest by changing the project browser view menu at the top to **Android**, and opening the `app/manifests` folder.
3. Add the `INTERNET`, `ACCESS_NETWORK_STATE`, and `ACCESS_WIFI_STATE`: permissions to your project's `AndroidManifest.xml` file.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.amazonaws.mobile.samples.mynotes">
```

```
    <uses-permission android:name="android.permission.INTERNET"/>
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
    <uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme"
        android:name=".Application">
        <!-- Other settings -->
    </application>
</manifest>
```

Add AWS SDK for Android library

1. **Edit the `app/build.gradle` file. Add the following lines to the**

   `dependencies` section:

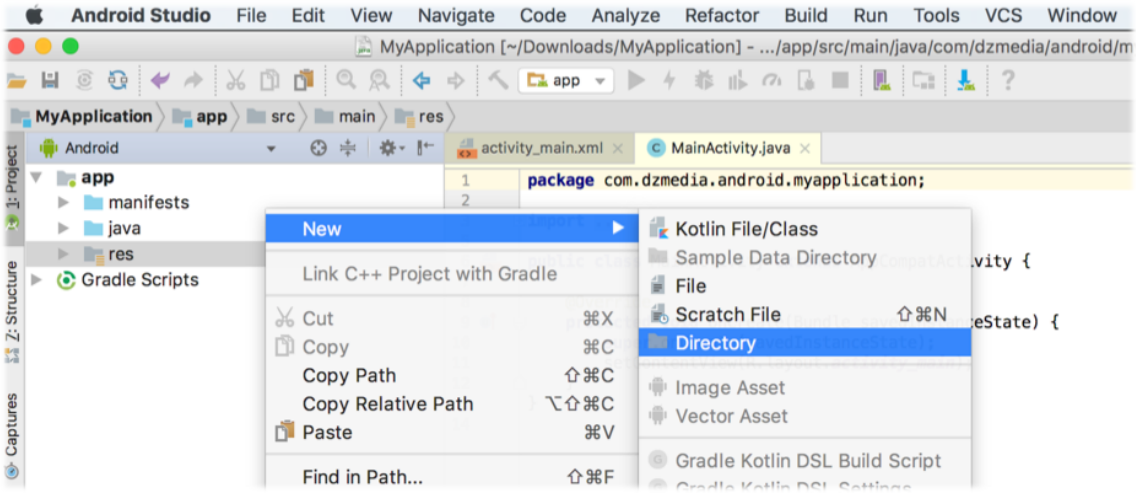
```
dependencies {
   compile fileTree(dir: 'libs', include: ['*.jar'])
   implementation 'com.android.support:appcompat-v7:26.1.0'
   implementation 'com.android.support:support-v4:26.1.0'
   implementation 'com.android.support:cardview-v7:26.1.0'
   implementation 'com.android.support:recyclerview-v7:26.1.0'
   implementation 'com.android.support.constraint:constraint-layout:1.0.2'
   implementation 'com.android.support:design:26.1.0'
   implementation 'com.android.support:multidex:1.0.1'
   implementation 'joda-time:joda-time:2.9.9'

   // AWS Mobile SDK for Android
   implementation 'com.amazonaws:aws-android-sdk-core:2.6.+'
   implementation 'com.amazonaws:aws-android-sdk-auth-core:2.6.+@aar'
   implementation 'com.amazonaws:aws-android-sdk-pinpoint:2.6.+'
}
```

2. Choose **Sync Now** on the upper right to incorporate the dependencies you just declared.

Integrate the AWS Configuration File

First, create a `raw` resource folder to store the AWS configuration file:

1. Expand the `app` folder.
2. Right-click the `res` folder.
3. Choose **New > Directory**.
4. Type `raw`.



5. Choose **OK**.
6. Copy the `awsconfiguration.json` file from its download location to the `app/src/main/res/raw` directory.

Android gives a resource ID to any arbitrary file placed in the `raw` folder, making it easy to reference in the app.

Tip	Use Reveal in Finder If you are having trouble locating the right directory on disk, use Android Studio. Right-click the `raw` folder, then select **Reveal in Finder**. A new window with the location of the `raw directory` pre-loaded will appear.

Create an AWSProvider.java Singleton Class

In our sample, all access to AWS is consolidated into a singleton class called `AWSProvider.java`.

1. Expand `app/java` in the Android Studio project explorer.
2. Right-click the `com.amazonaws.mobile.samples.mynotes` directory.
3. Select **New > Java Class**.
4. Enter the details:
   - Name: `AWSProvider`
   - Kind: `Singleton`
5. Choose **OK**.

You may be asked if you want to add the file to Git. Choose **Yes**.

The following is the initial code in this class:

```
package com.amazonaws.mobile.samples.mynotes;

import android.content.Context;

import com.amazonaws.auth.AWSCredentialsProvider;
```

```
import com.amazonaws.mobile.auth.core.IdentityManager;
import com.amazonaws.mobile.config.AWSConfiguration;
import com.amazonaws.mobileconnectors.pinpoint.PinpointConfiguration;
import com.amazonaws.mobileconnectors.pinpoint.PinpointManager;

public class AWSProvider {
    private static AWSProvider instance = null;
    private Context context;
    private AWSConfiguration awsConfiguration;
    private PinpointManager pinpointManager;

    public static AWSProvider getInstance() {
        return instance;
    }

    public static void initialize(Context context) {
        if (instance == null) {
            instance = new AWSProvider(context);
        }
    }

    private AWSProvider(Context context) {
        this.context = context;
        this.awsConfiguration = new AWSConfiguration(context);

        IdentityManager identityManager = new IdentityManager(context, awsConfiguration);
        IdentityManager.setDefaultIdentityManager(identityManager);
    }

    public Context getContext() {
        return this.context;
    }

    public AWSConfiguration getConfiguration() {
        return this.awsConfiguration;
    }

    public IdentityManager getIdentityManager() {
        return IdentityManager.getDefaultIdentityManager();
    }

    public PinpointManager getPinpointManager() {
        if (pinpointManager == null) {
            final AWSCredentialsProvider cp =
 getIdentityManager().getCredentialsProvider();
            PinpointConfiguration config = new PinpointConfiguration(
                    getContext(), cp, getConfiguration());
            pinpointManager = new PinpointManager(config);
        }
        return pinpointManager;
    }
}
```

What does this do?	The AWSProvider provides a central place to add code that accesses AWS resources. The constructor will load the AWS Configuration (a JSON file that you downloaded earlier) and create an IdentityManager object that is used to authenticate the device and/or user to AWS for accessing resources. The `getPinpointManager()` method will create a connection to Amazon Pinpoint if it doesn't exist.

Update the Application Class

All Android applications that include the AWS SDK for Android must inherit from MultiDexApplication. This has been done for you in this project. Open the `Application.java` file. In the `onCreate()` method of the `Application` class, add code to initialize the `AWSProvider` object we previously added:

```
public class Application extends MultiDexApplication {
   @Override
   public void onCreate() {
       super.onCreate();

       // Initialize the AWS Provider
       AWSProvider.initialize(getApplicationContext());

       registerActivityLifecycleCallbacks(new ActivityLifeCycle());
   }
}
```

Update the ActivityLifeCycle Class

We use an ActivityLifeCycle to monitor for activity events like start, stop, pause and resume. We need to determine when the user starts the application so that we can send a `startSession` event and `stopSession` event to Amazon Pinpoint. Adjust the `onActivityStarted()` and `onActivityStopped()` methods as follows:

```
@Override
public void onActivityStarted(Activity activity) {
    if (depth == 0) {
        Log.d("ActivityLifeCycle", "Application entered foreground");
        AWSProvider.getInstance().getPinpointManager().getSessionClient().startSession();
        AWSProvider.getInstance().getPinpointManager().getAnalyticsClient().submitEvents();
    }
    depth++;
}

@Override
public void onActivityStopped(Activity activity) {
    depth--;
    if (depth == 0) {
        Log.d("ActivityLifeCycle", "Application entered background");
        AWSProvider.getInstance().getPinpointManager().getSessionClient().stopSession();
        AWSProvider.getInstance().getPinpointManager().getAnalyticsClient().submitEvents();
    }
}
```

Monitor Add and Delete Events in Amazon Pinpoint

We can also monitor feature usage within our app. In this example, we will monitor how often users add and delete notes. We will record a custom event for each operation. The Delete Note operation occurs in the `NoteListActivity.java` class. Review the `onSwiped` method, and add the following code:

```
@Override
public void onSwiped(RecyclerView.ViewHolder viewHolder, int direction) {
    final NoteViewHolder noteHolder = (NoteViewHolder) viewHolder;
    ((NotesAdapter) notesList.getAdapter()).remove(noteHolder);

    // Send Custom Event to Amazon Pinpoint
    final AnalyticsClient mgr = AWSProvider.getInstance()
            .getPinpointManager()
            .getAnalyticsClient();
```

```
    final AnalyticsEvent evt = mgr.createEvent("DeleteNote")
            .withAttribute("noteId", noteHolder.getNote().getNoteId());
    mgr.recordEvent(evt);
    mgr.submitEvents();
}
```

The Add Note operation occurs in the `NoteDetailFragment.java` class. Review the `saveData()` method, and add code to send the custom event to Amazon Pinpoint as shown in the following fragment.

```
private void saveData() {
    // Save the edited text back to the item.
    boolean isUpdated = false;
    if (!mItem.getTitle().equals(editTitle.getText().toString().trim())) {
        mItem.setTitle(editTitle.getText().toString().trim());
        mItem.setUpdated(DateTime.now(DateTimeZone.UTC));
        isUpdated = true;
    }
    if (!mItem.getContent().equals(editContent.getText().toString().trim())) {
        mItem.setContent(editContent.getText().toString().trim());
        mItem.setUpdated(DateTime.now(DateTimeZone.UTC));
        isUpdated = true;
    }

    // Convert to ContentValues and store in the database.
    if (isUpdated) {
        ContentValues values = mItem.toContentValues();
        if (isUpdate) {
            contentResolver.update(itemUri, values, null, null);
        } else {
            itemUri = contentResolver.insert(NotesContentContract.Notes.CONTENT_URI,
 values);
            isUpdate = true;    // Anything from now on is an update

            // Send Custom Event to Amazon Pinpoint
            final AnalyticsClient mgr = AWSProvider.getInstance()
                    .getPinpointManager()
                    .getAnalyticsClient();
            final AnalyticsEvent evt = mgr.createEvent("AddNote")
                    .withAttribute("noteId", mItem.getNoteId());
            mgr.recordEvent(evt);
            mgr.submitEvents();
        }
    }
}
```

The AnalyticsClient and AnalyticsEvent classes are not imported by default. Use Alt-Return to import the missing classes.

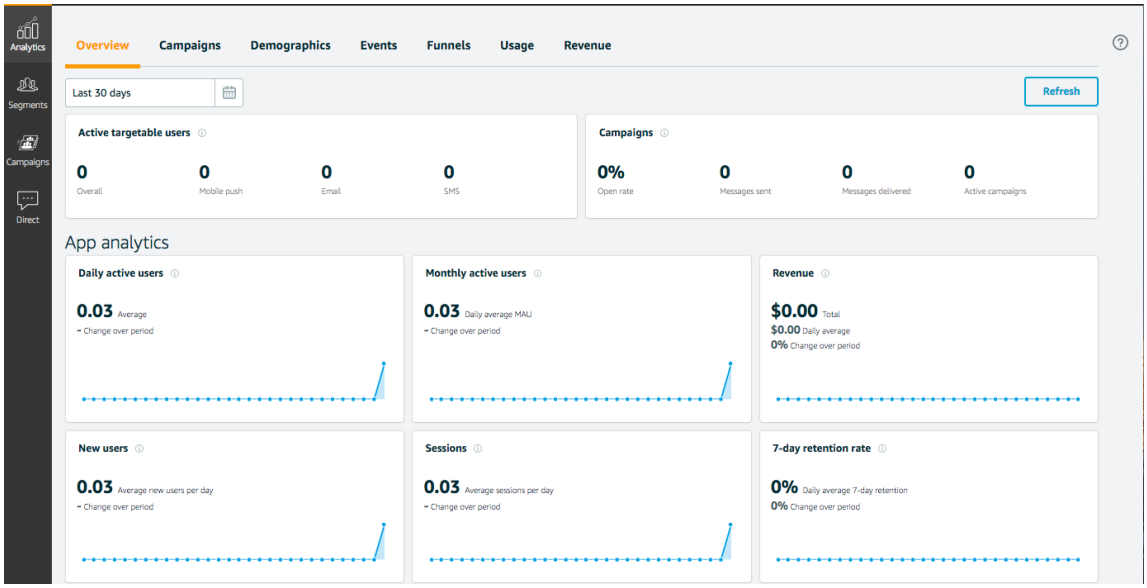
Tip	Auto Import You can set up Auto-Import to automatically import classes that you need. On Windows or Linux, you can find Auto-Import under **File > Settings**. On a Mac, you can find the same area under **Android Studio > Preferences**. The auto-import setting is under **Editor > General > Auto Import >Java**. Change **Insert imports on paste** to **All** and select the **Add unambiguous imports on the fly** option.

Run the Project and Validate Results

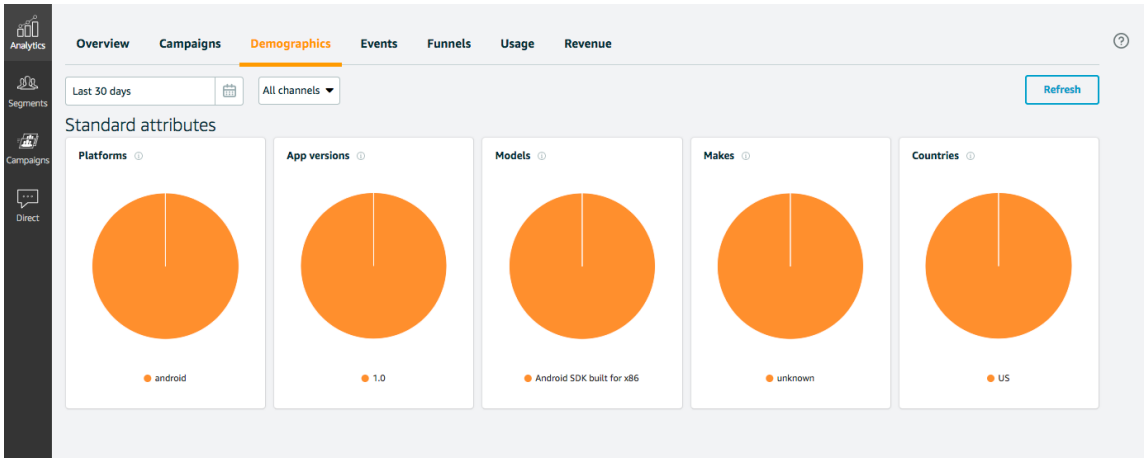
Run the application in the emulator using **Run** > **Run 'app'**. It should work as before. Ensure you try to add and delete some notes to generate some traffic that can be shown in the Pinpoint console.

To view the demographics and custom events:

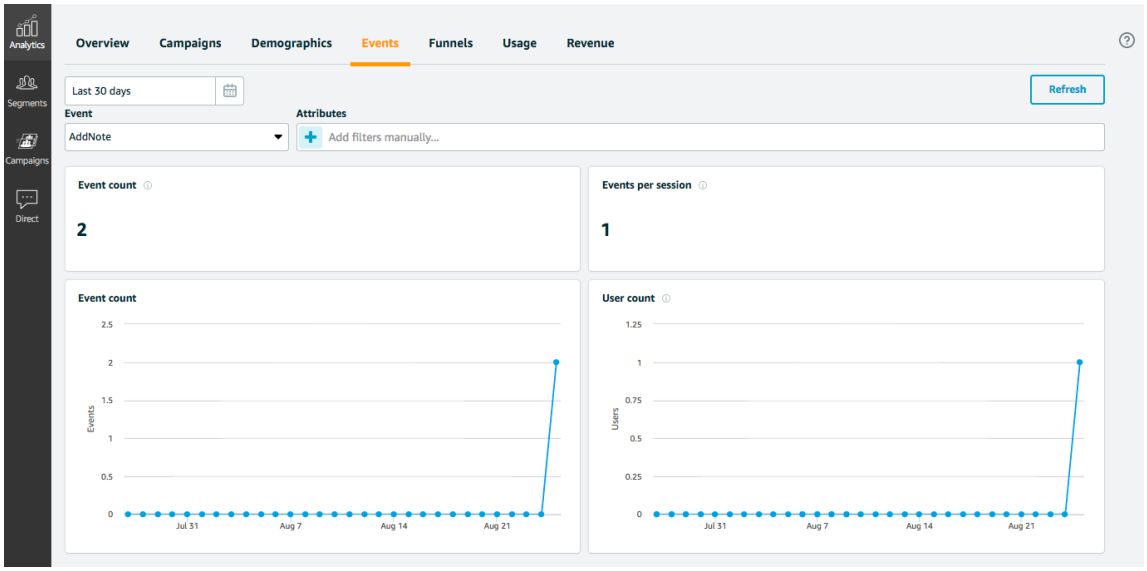
1. Open the AWS Mobile Hub console.
2. Choose your project.
3. Choose the **Analytics** icon on the left, to navigate to your project in the AWS Pinpoint console.
4. Choose **Analytics** on the left.
5. You should see an up-tick in several graphs:



6. Choose **Demographics** to view the demographics information.



7. Choose **Events**.
8. Use the Event drop down to show only the **AddNote** event.



If you see data within each page, you have successfully added analytics to your app. Should you release your app on the App Store, you can come back here to see more details about your users.

Next steps

- Continue by adding Authentication (p. 101).
- Learn more about Amazon Pinpoint.

Add Authentication to the Notes App

In the previous section (p. 94) of this tutorial, we created a mobile backend project in AWS Mobile Hub, then added analytics to the sample note-taking app. This section assumes you have completed those steps. If you jumped to this step, please go back and start from the beginning (p. 90). In this tutorial, we will configure a sign-up / sign-in flow in our mobile backend. We will then add a new authentication activity to our note-taking app.

You should be able to complete this section of the tutorial in 20-30 minutes.

Set Up Your Backend

Before we work on the client-side code, we need to add User Sign-in to the backend project:

1. Open the AWS Mobile Hub console.
2. Select your project.
3. Scroll down to the **Add More Backend Features** section.
4. Choose the **User Sign-in** tile.
5. Choose **Email and Password**.
6. Scroll to the bottom and then Choose **Create user pool**.

What does this do?	You have just created your own user pool in the Amazon Cognito service. When used in conjunction with the AWS Mobile sign-in process,

	the user pool enforces the password requirement rules you chose. It also supports sign-up and forgot my password user flows.

7. Choose your project name in the upper left and then choose **Integrate** on your Android app card.
8. Choose **Download Cloud Config** to get an `awsconfiguration.json` file updated with the new services.
9. Choose **Next** and then choose **Done**.

Remember	Whenever you update the AWS Mobile Hub project, a new AWS configuration file for your app is generated.

Connect to Your Backend

Replace the `awsconfiguration.json` file in `app/src/main/res/raw` directory with the updated version.

Note	Your system may have modified the filename to avoid conflicts. Make sure the file you add to your Xcode project is named `awsconfiguration.json`.

Add the Authentication UI Library

1. Open the `app/build.gradle` file and add the following lines to the `dependencies` section:

```
dependencies {
   compile fileTree(dir: 'libs', include: ['*.jar'])
   implementation 'com.android.support:appcompat-v7:26.1.0'
   implementation 'com.android.support:support-v4:26.1.0'
   implementation 'com.android.support:cardview-v7:26.1.0'
   implementation 'com.android.support:recyclerview-v7:26.1.0'
   implementation 'com.android.support.constraint:constraint-layout:1.0.2'
   implementation 'com.android.support:design:26.1.0'
   implementation 'com.android.support:multidex:1.0.1'
   implementation 'joda-time:joda-time:2.9.9'

   //AWS Mobile SDK for Android
   implementation 'com.amazonaws:aws-android-sdk-core:2.6.+'
   implementation 'com.amazonaws:aws-android-sdk-auth-core:2.6.+@aar'
   implementation 'com.amazonaws:aws-android-sdk-auth-ui:2.6.+@aar'
   implementation 'com.amazonaws:aws-android-sdk-auth-userpools:2.6.+@aar'
   implementation 'com.amazonaws:aws-android-sdk-cognitoidentityprovider:2.6.+'
   implementation 'com.amazonaws:aws-android-sdk-pinpoint:2.6.+'
 }
```

2. Choose **Sync Now** on the upper right to incorporate the dependencies you just declared.

Register the Email and Password Sign-in Provider

The sign-in UI is provided by `IdentityManager`. Each method of establishing identity (email and password, Facebook and Google) requires a plug-in provider that handles the appropriate sign-in flow.

1. Open your project in Android Studio.
2. Open the `AWSProvider.java` class.
3. Add the following to the import declarations:

```
import com.amazonaws.auth.AWSCredentialsProvider;
import com.amazonaws.mobile.auth.core.IdentityManager;
import com.amazonaws.mobile.auth.userpools.CognitoUserPoolsSignInProvider;
import com.amazonaws.mobile.config.AWSConfiguration;
import com.amazonaws.mobileconnectors.pinpoint.PinpointConfiguration;
import com.amazonaws.mobileconnectors.pinpoint.PinpointManager;
```

4. Adjust the constructor to add the `CognitoUserPoolsSignInProvider`.

```
private AWSProvider(Context context) {
   this.context = context;
   this.awsConfiguration = new AWSConfiguration(context);

   IdentityManager identityManager = new IdentityManager(context, awsConfiguration);
   IdentityManager.setDefaultIdentityManager(identityManager);
   identityManager.addSignInProvider(CognitoUserPoolsSignInProvider.class);
}
```

Add a AuthenticatorActivity to the project

You can call the IdentityProvider at any point in your application. In this tutorial, we will add a new screen to the project that is displayed before the list. The user will be prompted to sign-up or sign-in prior to seeing the list of notes. This ensures that all connections to the backend will be authenticated.

To add a AuthenticatorActivity to the project, in Android Studio

1. Right-click the `com.amazonaws.mobile.samples.mynotes` folder.
2. Choose **New > Activity > Empty Activity**.
3. Type `AuthenticatorActivity` as the **Activity Name**.
4. Choose **Finish**.

Edit the `onCreate()` method of `AuthenticatorActivity.java` as follows:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_authenticator);

    final IdentityManager identityManager = AWSProvider.getInstance().getIdentityManager();
    // Set up the callbacks to handle the authentication response
    identityManager.login(this, new DefaultSignInResultHandler() {
        @Override
        public void onSuccess(Activity activity, IdentityProvider identityProvider) {
            Toast.makeText(AuthenticatorActivity.this,
                    String.format("Logged in as %s", identityManager.getCachedUserID()),
                    Toast.LENGTH_LONG).show();
            // Go to the main activity
            final Intent intent = new Intent(activity, NoteListActivity.class)
                    .setFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
            activity.startActivity(intent);
            activity.finish();
        }
```

```
        @Override
        public boolean onCancel(Activity activity) {
            return false;
        }
    });

    // Start the authentication UI
    AuthUIConfiguration config = new AuthUIConfiguration.Builder()
            .userPools(true)
            .build();
    SignInActivity.startSignInActivity(this, config);
    AuthenticatorActivity.this.finish();
}
```

What does this do?	The AWS SDK for Android contains an in-built activity for handling the authentication UI. This Activity sets up the authentication UI to work for just email and password, then sets up an activity listener to handle the response. In this case, we transition to the `NoteListActivity` when a successful sign-in occurs, and stay on this activity when it fails. Finally, we transition to the Sign-In activity from the AWS SDK for Android library.

Update the AndroidManifest.xml

The `AuthenticatorActivity` will be added to the `AndroidManifest.xml` automatically, but it will not be set as the default (starting) activity. To make the AuthenticatorActivity primary, edit the `AndroidManifest.xml`:

```
<activity
    android:name=".AuthenticatorActivity"
    android:label="Sign In"
    android:theme="@style/AppTheme.NoActionBar">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
<activity
    android:name=".NoteListActivity"
    android:label="@string/app_name"
    android:theme="@style/AppTheme.NoActionBar">
    <!-- Remove the intent-filter from here -->
</activity>
```

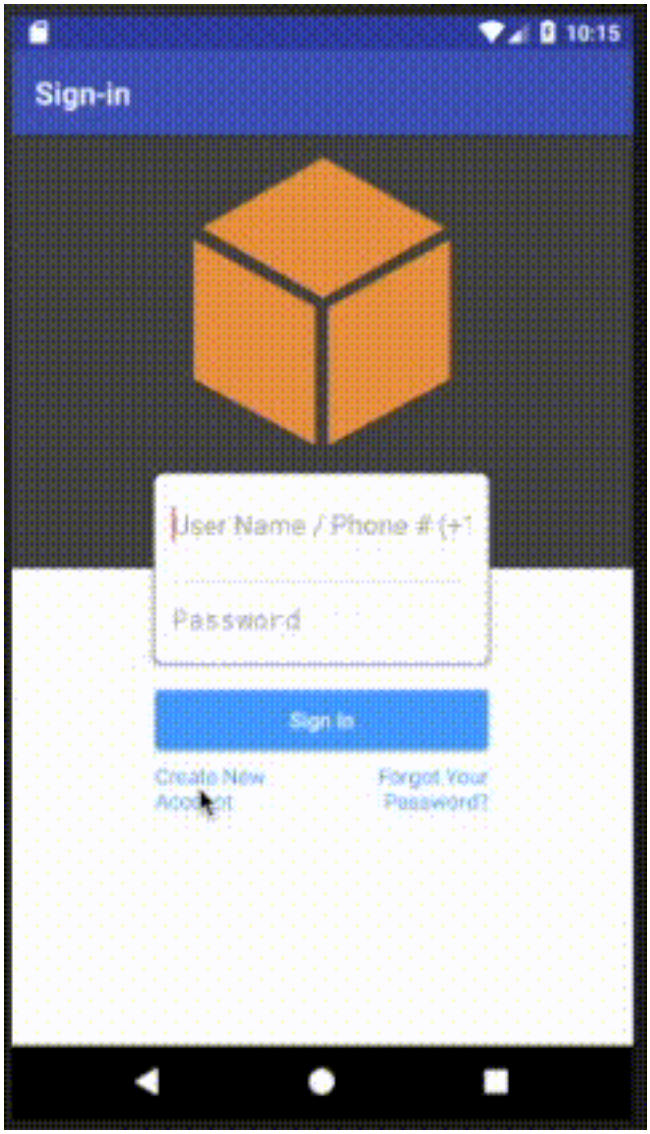
The `.AuthenticatorActivity` section is added at the end. Ensure it is not duplicated. You will see build errors if the section is duplicated.

Run the project and validate results

Run in the emulator using **Run** > **Run 'app'**. You should see a sign-in screen. Choose the **Create new account** button to create a new account. Once the information is submitted, you will be sent a confirmation code via email. Enter the confirmation code to complete registration, then sign-in with your new account.

Tip	Use Amazon WorkMail as a test email account

	If you do not want to use your own email account as a test account, create an Amazon WorkMail service within AWS for test accounts. You can get started for free with a 30-day trial for up to 25 accounts.



Next steps

- Continue by integrating NoSQL Data (p. 105).
- Learn more about Amazon Cognito.

Add Online Data Access to the Notes App

In the previous section (p. 101) of this tutorial , we added a simple sign-up / sign-in flow to the sample note-taking app with email validation. This tutorial assumes you have completed the previous tutorials.

If you jumped to this step, please go back and start from the beginning (p. 90). In this tutorial, we will add a NoSQL database to our mobile backend, then configure a basic data access service to the note-taking app.

The Notes sample app uses a ContentProvider (called `NotesContentProvider`) to provide access to a local SQLite database that is used to store the notes that you enter into the app. We will replace the code within the `ContentProvider` with code that uses DynamoDB instead of SQLite.

You should be able to complete this section of the tutorial in 30-45 minutes.

Add a NoSQL database to the AWS Mobile Hub project

Before we work on the client-side code, we need to add a NoSQL database and table to the backend project:

1. Open the AWS Mobile Hub console.
2. Select your project.
3. Scroll down to the **Add More Backend Features** section and then choose the **NoSQL Database** tile.
4. Choose **Enable NoSQL**, choose **Add Table**, and then choose **Example** to start with an example schema.
5. Choose **Notes**, which most closely matches the model we wish to use.
6. Choose **Add attribute**, then fill in the details of the new attribute:
   - **Attribute name**: `updatedDate`
   - **Type**: `number`
7. Choose **Add index** then fill in the details of the new index:
   - **Index name**: `LastUpdated`
   - **Partition key**: `userId`
   - **Sort key**: `updatedDate`
8. Choose **Create table**
9. Choose **Create table** in the modal dialog. It will take a few moments for AWS to create the table.

   You have just created a NoSQL table in the Amazon DynamoDB service.
10. When the table is ready, choose your project name in the upper left and then choose **Integrate** on your Android app card.
11. Choose **Download Cloud Config** to get an `awsconfiguration.json` file updated with the new services.
12. Choose **Next** and then choose **Done**.

Remember	Whenever you update the AWS Mobile Hub project, a new AWS configuration file for your app is generated.

Connect to Your Backend

Replace the `awsconfiguration.json` file in `app/src/main/res/raw` directory with the updated version.

Your system may have modified the filename to avoid conflicts. Make sure the file you add to your Android Studio project is named `awsconfiguration.json`.

Download the Models

To aid in implementing a provider for the table you created, Mobile Hub generated a data model descriptor file. To add the data model to your project:

1. Choose your project name in the upper left and then choose **Integrate** on the Android app card.
2. Choose **Android Models** under **Download Models**.
3. Unpack the downloaded ZIP file and copy the files under `src/main/java/com/amazonaws/models/nosql` to your Android Studio project in `app/src/main/java/com/amazonaws/mobile/samples/mynotes/data`. One file (`NotesDO.java`) should be copied.
4. Edit the `data/NotesDO.java` file and change the package setting:

```
package com.amazonaws.mobile.samples.mynotes.data;
```

Add required libraries to the project

Edit the `app/build.gradle` file and add the DynamoDB libraries to the dependencies:

```
dependencies {

    // . . .

    implementation 'com.amazonaws:aws-android-sdk-core:2.6.+'
    implementation 'com.amazonaws:aws-android-sdk-auth-core:2.6.+@aar'
    implementation 'com.amazonaws:aws-android-sdk-auth-ui:2.6.+@aar'
    implementation 'com.amazonaws:aws-android-sdk-auth-userpools:2.6.+@aar'
    implementation 'com.amazonaws:aws-android-sdk-cognitoidentityprovider:2.6.+'
    implementation 'com.amazonaws:aws-android-sdk-pinpoint:2.6.+'

    // Amazon DynamoDB for NoSQL tables
    implementation 'com.amazonaws:aws-android-sdk-ddb:2.6.+'
    implementation 'com.amazonaws:aws-android-sdk-ddb-mapper:2.6.+'
 }
```

1. Choose **Sync Now** on the upper right to incorporate the dependencies you just declared.

Add Data access methods to the AWSProvider class

To implement data synchronization, we need two explicit methods: a method to upload changes and a method to download updates from the server.

To add data access methods

1. Import `DynamoDBMapper` and `AmazonDynamoDBClient` in `AWSProvider.java`.

```
import com.amazonaws.auth.AWSCredentialsProvider;
import com.amazonaws.mobile.auth.core.IdentityManager;
import com.amazonaws.mobile.auth.userpools.CognitoUserPoolsSignInProvider;
import com.amazonaws.mobile.config.AWSConfiguration;
import com.amazonaws.mobile.samples.mynotes.data.NotesDO;
import com.amazonaws.mobileconnectors.pinpoint.PinpointConfiguration;
import com.amazonaws.mobileconnectors.pinpoint.PinpointManager;

// Add DynamoDBMapper and AmazonDynamoDBClient to support data access methods
import com.amazonaws.mobileconnectors.dynamodbv2.dynamodbmapper.DynamoDBMapper;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClient;
```

2. Add private `DynamoDBMapper` and `AmazonDynamoDBClient` variables to the `AWSProvider` class:

```
public class AWSProvider {
    private static AWSProvider instance = null;
    private Context context;
    private AWSConfiguration awsConfiguration;
    private PinpointManager pinpointManager = null;

    // Declare DynamoDBMapper and AmazonDynamoDBClient private variables
    // to support data access methods
    private AmazonDynamoDBClient dbClient = null;
    private DynamoDBMapper dbMapper = null;

    public static AWSProvider getInstance() {
          return instance;
    }
}
```

3. Add the following method to the class:

```
public DynamoDBMapper getDynamoDBMapper() {
    if (dbMapper == null) {
        final AWSCredentialsProvider cp = getIdentityManager().getCredentialsProvider();
        dbClient = new AmazonDynamoDBClient(cp);
        dbMapper = DynamoDBMapper.builder()
                .awsConfiguration(getConfiguration())
                .dynamoDBClient(dbClient)
                .build();
    }
    return dbMapper;
}
```

Implement Mutation Methods

The ContentProvider is the basic interface that Android uses to communicate with databases on Android. It uses four methods that match the basic CRUD (create, read, update, delete) methods.

Add the following methods to the `NotesContentProvider` class:

```
private NotesDO toNotesDO(ContentValues values) {
    final NotesDO note = new NotesDO();
    note.setContent(values.getAsString(NotesContentContract.Notes.CONTENT));
    note.setCreationDate(values.getAsDouble(NotesContentContract.Notes.CREATED));
    note.setNoteId(values.getAsString(NotesContentContract.Notes.NOTEID));
    note.setTitle(values.getAsString(NotesContentContract.Notes.TITLE));
    note.setUpdatedDate(values.getAsDouble(NotesContentContract.Notes.UPDATED));
    note.setUserId(AWSProvider.getInstance().getIdentityManager().getCachedUserID());
    return note;
}

private Object[] fromNotesDO(NotesDO note) {
    String[] fields = NotesContentContract.Notes.PROJECTION_ALL;
    Object[] r = new Object[fields.length];
    for (int i = 0 ; i < fields.length ; i++) {
        if (fields[i].equals(NotesContentContract.Notes.CONTENT)) {
            r[i] = note.getContent();
        } else if (fields[i].equals(NotesContentContract.Notes.CREATED)) {
            r[i] = note.getCreationDate();
        } else if (fields[i].equals(NotesContentContract.Notes.NOTEID)) {
            r[i] = note.getNoteId();
        } else if (fields[i].equals(NotesContentContract.Notes.TITLE)) {
            r[i] = note.getTitle();
```

```
        } else if (fields[i].equals(NotesContentContract.Notes.UPDATED)) {
            r[i] = note.getUpdatedDate();
        } else {
            r[i] = new Integer(0);
        }
    }
    return r;
}
```

These functions convert object attributes when they are passed between `ContentValues` of the app and the `NotesDO` object, which required by the Amazon DynamoDB service.

Mutation events handle the `insert`, `update`, and `delete` methods:

```
@Nullable
@Override
public Uri insert(@NonNull Uri uri, @Nullable ContentValues values) {
    int uriType = sUriMatcher.match(uri);
    switch (uriType) {
        case ALL_ITEMS:
            DynamoDBMapper dbMapper = AWSProvider.getInstance().getDynamoDBMapper();
            final NotesDO newNote = toNotesDO(values);
            dbMapper.save(newNote);
            Uri item = NotesContentContract.Notes.uriBuilder(newNote.getNoteId());
            notifyAllListeners(item);
            return item;
        default:
            throw new IllegalArgumentException("Unsupported URI: " + uri);
    }
}

@Override
public int delete(@NonNull Uri uri, @Nullable String selection, @Nullable String[]
 selectionArgs) {
    int uriType = sUriMatcher.match(uri);
    int rows;

    switch (uriType) {
        case ONE_ITEM:
            DynamoDBMapper dbMapper = AWSProvider.getInstance().getDynamoDBMapper();
            final NotesDO note = new NotesDO();
            note.setNoteId(uri.getLastPathSegment());

 note.setUserId(AWSProvider.getInstance().getIdentityManager().getCachedUserID());
            dbMapper.delete(note);
            rows = 1;
            break;
        default:
            throw new IllegalArgumentException("Unsupported URI: " + uri);
    }
    if (rows > 0) {
        notifyAllListeners(uri);
    }
    return rows;
}

@Override
public int update(@NonNull Uri uri, @Nullable ContentValues values, @Nullable String
 selection, @Nullable String[] selectionArgs) {
    int uriType = sUriMatcher.match(uri);
    int rows;

    switch (uriType) {
        case ONE_ITEM:
```

```
            DynamoDBMapper dbMapper = AWSProvider.getInstance().getDynamoDBMapper();
            final NotesDO updatedNote = toNotesDO(values);
            dbMapper.save(updatedNote);
            rows = 1;
            break;
        default:
            throw new IllegalArgumentException("Unsupported URI: " + uri);
    }
    if (rows > 0) {
        notifyAllListeners(uri);
    }
    return rows;
}
```

Implement Query Methods

This application always asks for the entire data set that the user is entitled to see, so there is no need to implement complex query management. This simplifies the `query()` method considerably. The `query()` method returns a `Cursor` (which is a standard mechanism for iterating over data sets returned from databases).

```
@Nullable
@Override
public Cursor query(
              @NonNull Uri uri,
              @Nullable String[] projection,
              @Nullable String selection,
              @Nullable String[] selectionArgs,
              @Nullable String sortOrder) {
    int uriType = sUriMatcher.match(uri);

    DynamoDBMapper dbMapper = AWSProvider.getInstance().getDynamoDBMapper();
    MatrixCursor cursor = new MatrixCursor(NotesContentContract.Notes.PROJECTION_ALL);
    String userId = AWSProvider.getInstance().getIdentityManager().getCachedUserID();

    switch (uriType) {
        case ALL_ITEMS:
            // In this (simplified) version of a content provider, we only allow searching
            // for all records that the user owns.  The first step to this is establishing
            // a template record that has the partition key pre-populated.
            NotesDO template = new NotesDO();
            template.setUserId(userId);
            // Now create a query expression that is based on the template record.
            DynamoDBQueryExpression<NotesDO> queryExpression;
            queryExpression = new DynamoDBQueryExpression<NotesDO>()
                    .withHashKeyValues(template);
            // Finally, do the query with that query expression.
            List<NotesDO> result = dbMapper.query(NotesDO.class, queryExpression);
            Iterator<NotesDO> iterator = result.iterator();
            while (iterator.hasNext()) {
                final NotesDO note = iterator.next();
                Object[] columnValues = fromNotesDO(note);
                cursor.addRow(columnValues);
            }

            break;
        case ONE_ITEM:
            // In this (simplified) version of a content provider, we only allow searching
            // for the specific record that was requested
            final NotesDO note = dbMapper.load(NotesDO.class, userId,
 uri.getLastPathSegment());
            if (note != null) {
                Object[] columnValues = fromNotesDO(note);
                cursor.addRow(columnValues);
```

```
            }
            break;
    }

    cursor.setNotificationUri(getContext().getContentResolver(), uri);
    return cursor;
}
```

Note	Differences from a real implementation We've taken a simplified approach for this content provider to demonstrate the CRUD implementation. A real implementation would need to deal with online state and handle caching of the data, plus handle appropriate query capabilities as required by the application.

Convert the CRUD methods to Async

The in-built SQLite driver has asynchronous wrappers so that you don't need to think about what the content provider is actually doing. However, network connections cannot happen on the UI thread. In the absence of an asynchronous wrapper, you must provide your own. This affects the create, update, and delete operations. There is no need to add code to load the data from the server, as that operation is already asynchronous.

Inserts and updates are done in the `NoteDetailFragment.java` class. Deletes are done in the `NoteListActivity.java` class.

In the `OnCreate()` method of the `NoteDetailFragment.java` class, replace the following `if` statement that calls local cursor functions:

```
if (arguments != null && arguments.containsKey(ARG_ITEM_ID)) {
    String itemId = getArguments().getString(ARG_ITEM_ID);
    itemUri = NotesContentContract.Notes.uriBuilder(itemId);
    Cursor data = contentResolver.query(itemUri, NotesContentContract.Notes.PROJECTION_ALL,
 null, null, null);
    if (data != null) {
        data.moveToFirst();
        mItem = Note.fromCursor(data);
        isUpdate = true;
    }
} else {
    mItem = new Note();
    isUpdate = false;
}
```

With the following constants and statement that establishes an `AsyncQueryHandler`, which provides a wrapper to make the calls run on a non-UI thread asynchronously:

```
// Constants used for async data operations
private static final int QUERY_TOKEN = 1001;
private static final int UPDATE_TOKEN = 1002;
private static final int INSERT_TOKEN = 1003;

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
```

```
    // Get the ContentResolver
    contentResolver = getContext().getContentResolver();

    // Unbundle the arguments if any.  If there is an argument, load the data from
    // the content resolver aka the content provider.
    Bundle arguments = getArguments();
    mItem = new Note();
    if (arguments != null && arguments.containsKey(ARG_ITEM_ID)) {
        String itemId = getArguments().getString(ARG_ITEM_ID);
        itemUri = NotesContentContract.Notes.uriBuilder(itemId);


        // Replace local cursor methods with async query handling
        AsyncQueryHandler queryHandler = new AsyncQueryHandler(contentResolver) {
            @Override
            protected void onQueryComplete(int token, Object cookie, Cursor cursor) {
                super.onQueryComplete(token, cookie, cursor);
                cursor.moveToFirst();
                mItem = Note.fromCursor(cursor);
                isUpdate = true;

                editTitle.setText(mItem.getTitle());
                editContent.setText(mItem.getContent());
            }
        };
        queryHandler.startQuery(QUERY_TOKEN, null, itemUri,
 NotesContentContract.Notes.PROJECTION_ALL, null, null, null);


    } else {
        isUpdate = false;
    }

    // Start the timer for the delayed start
    timer.postDelayed(timerTask, 5000);
}
```

In the `saveData()` method, replace the following local cursor methods:

```
// Convert to ContentValues and store in the database.
if (isUpdated) {
    ContentValues values = mItem.toContentValues();
    if (isUpdate) {
        contentResolver.update(itemUri, values, null, null);
    } else {
        itemUri = contentResolver.insert(NotesContentContract.Notes.CONTENT_URI, values);
        isUpdate = true;    // Anything from now on is an update
        itemUri = NotesContentContract.Notes.uriBuilder(mItem.getNoteId());
    }
}
```

with an `AsyncQueryHandler`:

```
private void saveData() {
    // Save the edited text back to the item.
    boolean isUpdated = false;
    if (!mItem.getTitle().equals(editTitle.getText().toString().trim())) {
        mItem.setTitle(editTitle.getText().toString().trim());
        mItem.setUpdated(DateTime.now(DateTimeZone.UTC));
        isUpdated = true;
    }
    if (!mItem.getContent().equals(editContent.getText().toString().trim())) {
```

```
        mItem.setContent(editContent.getText().toString().trim());
        mItem.setUpdated(DateTime.now(DateTimeZone.UTC));
        isUpdated = true;
    }

    // Replace local cursor methods with an async query handler
    // Convert to ContentValues and store in the database.
    if (isUpdated) {
        ContentValues values = mItem.toContentValues();

        AsyncQueryHandler queryHandler = new AsyncQueryHandler(contentResolver) {
            @Override
            protected void onInsertComplete(int token, Object cookie, Uri uri) {
                super.onInsertComplete(token, cookie, uri);
                Log.d("NoteDetailFragment", "insert completed");
            }

            @Override
            protected void onUpdateComplete(int token, Object cookie, int result) {
                super.onUpdateComplete(token, cookie, result);
                Log.d("NoteDetailFragment", "update completed");
            }
        };
        if (isUpdate) {
            queryHandler.startUpdate(UPDATE_TOKEN, null, itemUri, values, null, null);
        } else {
            queryHandler.startInsert(INSERT_TOKEN, null,
 NotesContentContract.Notes.CONTENT_URI, values);
            isUpdate = true;    // Anything from now on is an update

            // Send Custom Event to Amazon Pinpoint
            final AnalyticsClient mgr = AWSProvider.getInstance()
                    .getPinpointManager()
                    .getAnalyticsClient();
            final AnalyticsEvent evt = mgr.createEvent("AddNote")
                    .withAttribute("noteId", mItem.getNoteId());
            mgr.recordEvent(evt);
            mgr.submitEvents();
        }


    }
}
```

Replace the `remove()` method in `NoteListActivity.java` with the following.

```
private static final int DELETE_TOKEN = 1004;

void remove(final NoteViewHolder holder) {
    if (mTwoPane ){
        // Check to see if the current fragment is the record we are deleting
        Fragment currentFragment =
 NoteListActivity.this.getSupportFragmentManager().findFragmentById(R.id.note_detail_container);
        if (currentFragment instanceof NoteDetailFragment) {
            String deletedNote = holder.getNote().getNoteId();
            String displayedNote = ((NoteDetailFragment)
 currentFragment).getNote().getNoteId();
            if (deletedNote.equals(displayedNote)) {

 getSupportFragmentManager().beginTransaction().remove(currentFragment).commit();
            }
        }
    }

    // Remove the item from the database
```

```
    final int position = holder.getAdapterPosition();
    Uri itemUri = NotesContentContract.Notes.uriBuilder(holder.getNote().getNoteId());
    AsyncQueryHandler queryHandler = new AsyncQueryHandler(getContentResolver()) {
        @Override
        protected void onDeleteComplete(int token, Object cookie, int result) {
            super.onDeleteComplete(token, cookie, result);
            notifyItemRemoved(position);
            Log.d("NoteListActivity", "delete completed");
        }
    };

    queryHandler.startDelete(DELETE_TOKEN, null, itemUri, null, null);
}
```

If you need to do a query (for example, to respond to a search request), then you can use a similar technique to wrap the `query()` method.

Run the application

You must be online in order to run this application. Run the application in the emulator. Note that the initial startup after logging in is slightly longer (due to reading the data from the remote database).

Data is available immediately in the mobile backend. Create a few notes, then view the records within the AWS Console:

1. Open the Mobile Hub console.
2. Choose your project.
3. Choose **Resources** in the left hand menu.
4. Choose the link for your DynamoDB table.
5. Choose the **Items** tab.

When you insert, edit or delete notes in the app, you should be able to see the data on the server reflect your actions almost immediately.

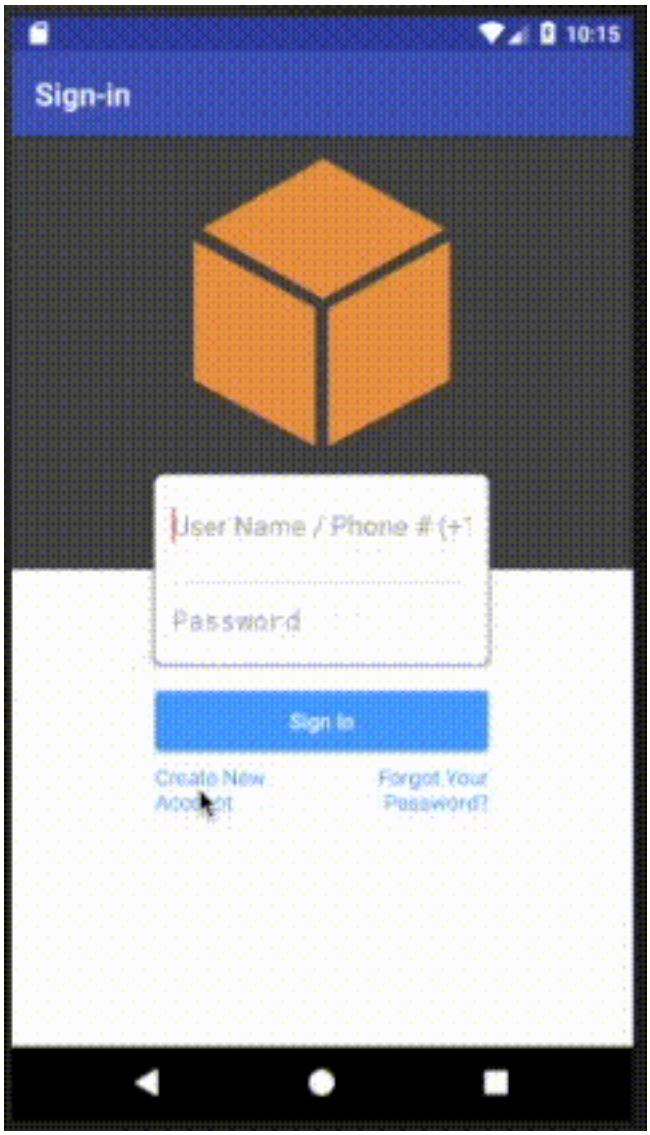
Next Steps

- Learn about data synchronization by reading about the Android Sync Framework.
- Learn about Amazon DynamoDB.

A Simple Note-taking App

Start with a working app and then add cloud enable features. In this tutorial, you will take a working app, driven from locally stored data, and then:

- Add analytics to your app (p. 116) so you can view demographic information about your users.
- Add a simple sign-in/sign-up flow (p. 121) for authentication.
- Access data stores in the AWS (p. 126) cloud, so that a user's notes are available to them on any device with the app installed.



You should be able to complete the setup section of this tutorial within 10-15 minutes after you have installed all required software. Once you complete the instructions on this page, you can run the project on your local system.

Getting Started

Before beginning, on your Mac:

- Install XCode using the Mac App Store (version 8.0 or higher is required).
- Configure the XCode command line tools. Run `xcode-select --install` from a Terminal window.
- Install Cocoapods. From a terminal window run:

```
sudo gem install cocoapods
```

Download the Source code

1. Get the tutorial source code using one of the following choices:

- Download the source code as a ZIP file.
- Browse to https://github.com/aws-samples/aws-mobile-ios-notes-tutorial/ and clone or fork the repository (sign up for GitHub account, if you do not have one).

Compile and Run the Project

To compile the source code and the project in a simulator:

1. Unzip `aws-mobile-ios-notes-tutorial-latest.zip` and launch Xcode by choosing `MyNotes.xcodeproj` in the expanded folder.
2. Select **Product > Build** (
<problematic>|Acommand|</problematic>
-B) to build the project.
3. Select any compatible simulator from the list in the toolbar at the top, next to the label with your app name.
4. Choose the run icon (
<problematic>|play|</problematic>
) on the top left or type
<problematic>|Acommand|</problematic>
-R to build and run the app.

You should be able to interact with the application. Try clicking on the **+** at the top right to create a note, or click on a note to show the editor screen.

Next Steps

Next, integrate application analytics (p. 116) into your project.

Add Analytics to the Notes App

In the previous section (p. 114) of this tutorial, we installed Xcode, downloaded a sample note-taking app from GitHub, then compiled and ran it in the iOS Simulator. This tutorial assumes you have completed the those steps. In this section, we will extend the notes app to include application analytics. Application analytics allow us to gather demographic information about the application usage.

You should be able to complete this section in 10-15 minutes.

Set Up Your Back End

To start, set up the mobile backend resources in AWS:

1. Open the AWS Mobile Hub console.
   - If you do not have an AWS account, sign up for the AWS Free Tier.
2. Choose **Create** on the upper left, and the type `ios-notes-app` for the name of the Mobile Hub project.
3. Choose **Next**, choose **iOS**, and then choose **Add**.
4. Choose **Download Cloud Config**, and save `awsconfiguration.json`. This file the configuration to connect your app to your backend.
5. Choose **Next** and then choose **Done** to create the project.

Used in this section	AWS Mobile Hub: Configure your mobile app's AWS backend in minutes, and then to manage those resources as your app evolves.

Connect to Your Backend

1. Drag `awsconfiguration.json` from the download location into the folder in the XCode Project Navigator that contains `Info.plist`. Select **Copy items if needed** and **Create groups** in the options dialog.
2. Choose **Finish**.

You have now created the AWS resources you need and connected them to your app.

Add Analytics the Dependencies

1. To create a `Podfile` for your project, run:

```
cd YOUR-APP-ROOT-FOLDER
pod init
```

2. Open `Podfile` and replace the placeholder code with the following. If the file is not visible your Xcode Project Navigator, right-click the project root and choose **Show in finder**.

```
platform :ios, '9.0'
target :'MyNotes' do
  use_frameworks!

    # Analytics dependency
    pod 'AWSPinpoint', '~> 2.6.5'

    # other pods

end
```

3. Close your Xcode project and then run:

```
pod install --repo-update
```

If you encounter an error message that begins "`[!] Failed to connect to GitHub to update the CocoaPods/Specs . . .`", and your internet connectivity is working, you may need to update openssl and Ruby.

Important	From this point forward, **open your project using the `.xcworkspace` file** generated by cocoapods for all further development.

4. Rebuild your app after reopening it in the workspace to resolve APIs from new libraries called in your code. This is a good practice any time you add import statements.

Initialize Amazon Pinpoint to Enable Analytics

You have just installed the AWS Mobile dependencies for your app.

To turn your analytics on, open your project using `MyNotes.xcworkspace` insert the following code into the `didFinishLaunchwithOptions` method of your app's `AppDelegate.swift`.

```
             //. . .

// Analytics imports
import AWSCore
import AWSPinpoint

             //. . .


 class AppDelegate: UIResponder, UIApplicationDelegate {

             //. . .

     // Add the pinpoint variable
     var pinpoint: AWSPinpoint?

             //. . .

     func application(_ application: UIApplication, didFinishLaunchingWithOptions
 launchOptions:
     [UIApplicationLaunchOptionsKey: Any]?) -> Bool {

             //. . .

        // Initialize Pinpoint to enable session analytics
        pinpoint = AWSPinpoint(configuration:
             AWSPinpointConfiguration.defaultPinpointConfiguration(launchOptions:
 launchOptions))

        return true
     }

             //. . .

 }
```

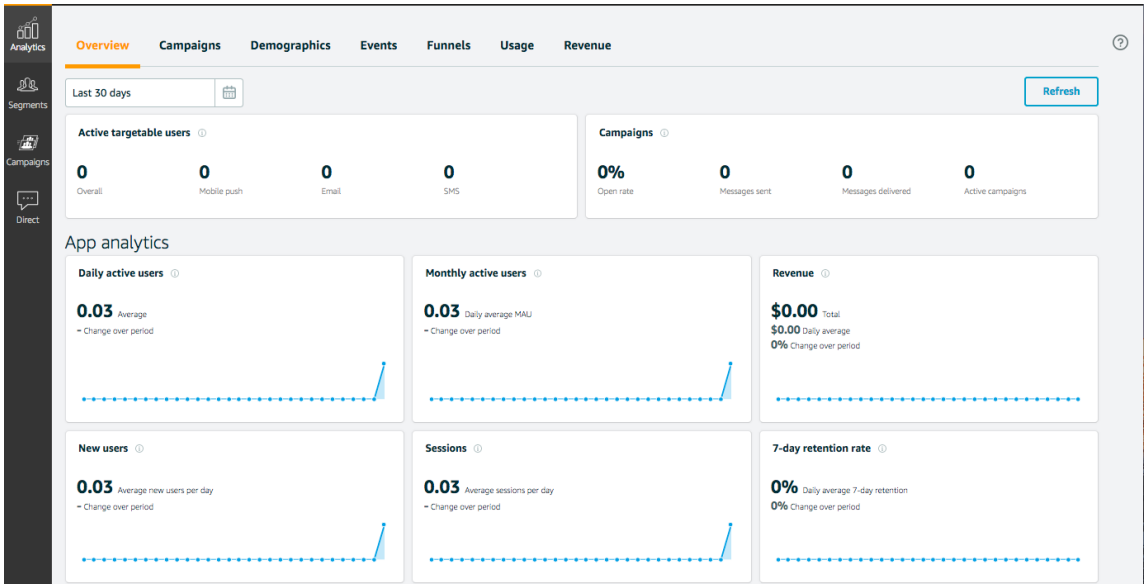
Now your app is setup to provide session analytics you can view in the Amazon Pinpoint console.

Run the App and Validate Results

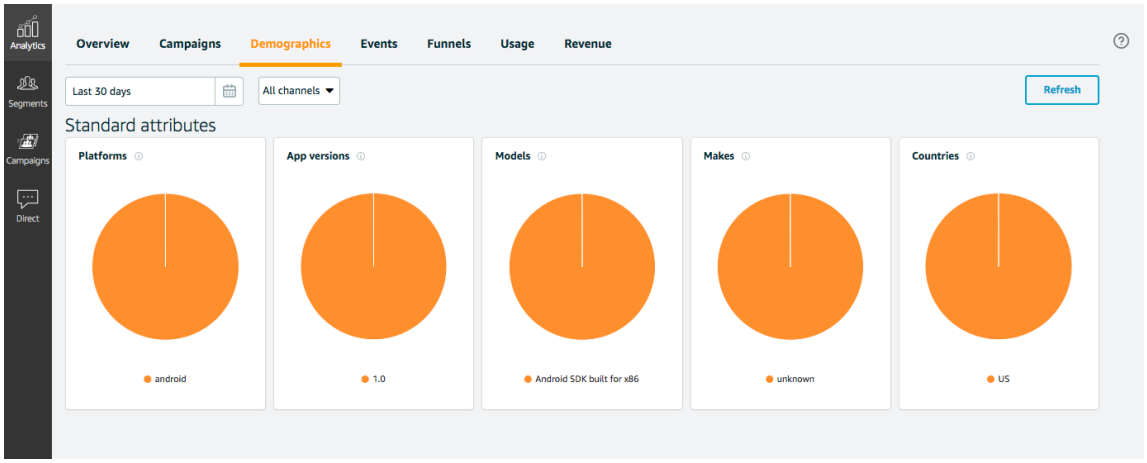
Re-build the application and run the application in the Simulator. It should work as before. Add and delete some notes to generate analytics traffic that can be shown in the Pinpoint console.

To view the demographics and custom events:

1. Choose **Analytics** on the top right to open your project in the Amazon Pinpoint console.
2. Choose the **Analytics** icon on the left. You should see an up-tick in several graphs (it may take a few minutes for the data to show):



3. Choose **Demographics** to view the demographics information.



If you see data within each page, you have successfully added analytics to your app. Should you release your app, you can come back here to see more details about your users.

Add Custom Analytics

Amazon Pinpoint also allows you to capture custom analytics data for events that show usage of your app's features. For this tutorial we'll send analytics each time a note is added or deleted.

To add analytics events, open `./Data/NotesContentProvider.swift` where both add and delete operations happen.

Start by adding the following imports.

```
import AWSCore
import AWSPinpoint
```

Add the following function and enum to the `NotesContentProvider` class to send `AddNote` and `DeleteNote` event analytics.

```
public class NotesContentProvider  {

         // . . .


    // Send analytics AddNote and DeleteNote events
    func sendNoteEvent(noteId: String, eventType: String)
    {

        let pinpointClient = AWSPinpoint(configuration:
            AWSPinpointConfiguration.defaultPinpointConfiguration(launchOptions: nil))

        let pinpointAnalyticsClient = pinpointClient.analyticsClient

        let event = pinpointAnalyticsClient.createEvent(withEventType: eventType)
        event.addAttribute("NoteId", forKey: noteId)
        pinpointAnalyticsClient.record(event)
        pinpointAnalyticsClient.submitEvents()
    }

    enum noteEventType: String {
        case AddNote = "AddNote"
        case DeleteNote = "DeleteNote"
    }

}
```

To capture note additions, place the following `sendNoteEvent` function call within the `insert` function of that class.

```
 /**
  * Insert a new record into the database using NSManagedObjectContext
  *
  * @param noteTitle the note title to be inserted
  * @param noteContent the note content to be inserted
  * @return noteId the unique Note Id
  */
func insert(noteTitle: String, noteContent: String) -> String {

            // . . .

     print("New Note Saved : \(newNoteId)")

     //Send AddNote analytics event
     sendNoteEvent(noteId: newNoteId, eventType: noteEventType.AddNote.rawValue)

     return newNoteId
}
```

To capture note deletions, place the following `sendNoteEvent` function call within the `delete` function of that class.

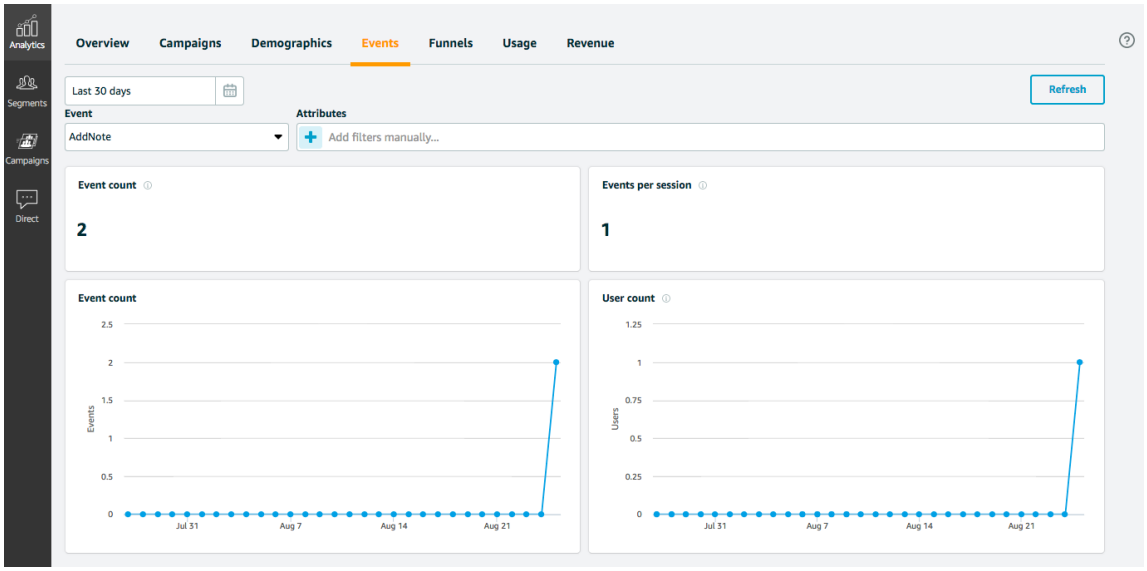
```
 /**
 * Delete note using NSManagedObjectContext and NSManagedObject
 * @param managedObjectContext the managed context for the note to be deleted
 * @param managedObj the core data managed object for note to be deleted
 * @param noteId the noteId to be delete
 */
public func delete(managedObjectContext: NSManagedObjectContext, managedObj:
 NSManagedObject, noteId: String!)  {
    let context = managedObjectContext
    context.delete(managedObj)
```

```
    do {

            // . . .

        // Send DeletNote analytics event
        sendNoteEvent(noteId: noteId, eventType: noteEventType.DeleteNote.rawValue)

            // . . .

    } catch {

            // . . .
    }
}
```

View Your Custom Analytics

To view the `AddNote` and `DeleteNote` custom analytics events, rebuild and run your app in the Simulator, add and delete notes, then return to the Amazon Pinpoint console for your project.

1. Choose **Events**.
2. Use the Event drop down to filter the event type (event types may take several minutes to appear).



Next steps

- Continue by adding Authentication (p. 121).
- Learn more about Amazon Pinpoint.

Add Authentication to the Notes App

In the previous section (p. 116) of this tutorial, we created a mobile backend project in AWS Mobile Hub, then added analytics to the sample note-taking app. This section assumes you have completed those steps. If you jumped to this step, please go back and start from the beginning (p. 114). In this tutorial, we will configure a sign-up / sign-in flow in our mobile backend. We will then add a new authentication activity to our note-taking app.

You should be able to complete this section of the tutorial in 20-30 minutes.

Setup Your Backend

To add User Sign-in to your app you will create the backend resources in your Mobile Hub project, and then update the configuration file in your app.

Add User Sign-in to the AWS Mobile Hub Project

1. Right-click `awsconfiguration.json` in your Xcode Project Navigator, choose **Delete**, and then choose **Move to trash**.
2. Open the AWS Mobile Hub console.
3. Select your project.
4. Scroll down to the **Add More Backend Features** section.
5. Choose the **User Sign-in** tile.
6. Choose **Email and Password**.
7. Scroll to the bottom and then Choose **Create user pool**.

What does this do?	You have just created your own user pool in the Amazon Cognito service. When used in conjunction with the AWS Mobile sign-in process, the user pool enforces the password requirement rules you chose. It also supports sign-up and forgot my password user flows.

8. Choose your project name in the upper left and then choose **Integrate** on your iOS app card.
9. Choose **Download Cloud Config** to get an `awsconfiguration.json` file updated with the new services.

Remember	Whenever you update the AWS Mobile Hub project, a new AWS configuration file for your app is generated.

Connect to Your Backend

To update the linkage between your app and your AWS services:

1. Drag `awsconfiguration.json` from your download location into the Xcode project folder containing `Info.plist`. Select **Copy items if needed** and **Create groups** in the options dialog. Choose **Finish**.

Note	Your system may have modified the filename to avoid conflicts. Make sure the file you add to your Xcode project is named `awsconfiguration.json`.

Add Auth Dependencies

1. Add the following Auth dependencies in your project's `Podfile`

```
platform :ios, '9.0'
```

```
target :'MyNotes' do
    use_frameworks!

      # Analytics dependency
      pod 'AWSPinpoint', '~> 2.6.5'

      # Auth dependencies
      pod 'AWSUserPoolsSignIn', '~> 2.6.5'
      pod 'AWSAuthUI', '~> 2.6.5'
      pod 'AWSMobileClient', '~> 2.6.5'

    # other pods
end
```

Then, in a terminal run:

```
pod install --repo-update
```

If you encounter an error message that begins "`[!] Failed to connect to GitHub to update the CocoaPods/Specs . . .`", and your internet connectivity is working, you may need to update openssl and Ruby.

Create an AWSMobileClient and Initialize the SDK

Import `AWSMobileClient` and add the following function into the `AppDelegate` class of `AppDelegate.swift`. This will create an instance of `AWSMobileClient`.

```
import UIKit
import CoreData

// Anaytics imports
import AWSCore
import AWSPinpoint

// Auth imports
import AWSMobileClient

@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate {

               // . . .

    //Instantiate the AWSMobileClient
    func application(_ application: UIApplication, open url: URL,
        sourceApplication: String?, annotation: Any) -> Bool {

        return AWSMobileClient.sharedInstance().interceptApplication(
            application, open: url,
            sourceApplication: sourceApplication,
            annotation: annotation)

    }

                // . . .
}
```

In `didFinishLaunching` call the `AWSMobileClient` to register your user pool as the identity provider that enables users to access your app's AWS resources.

```
 func application(
```

```
    _ application: UIApplication,
    didFinishLaunchingWithOptions launchOptions:
    [UIApplicationLaunchOptionsKey: Any]?) -> Bool {

    // Initialize AWSMobileClient
    let didFinishLaunching = AWSMobileClient.sharedInstance().interceptApplication(
        application, didFinishLaunchingWithOptions:
        launchOptions)

    // Initialize Pinpoint to enable session analytics
    let pinpoint = AWSPinpoint(configuration:
        AWSPinpointConfiguration.defaultPinpointConfiguration(
            launchOptions: launchOptions))

    return didFinishLaunching
}
```

What did this do?	This will register your sign in providers and fetch the user pool you created and fetch an identity that enables a user to access your app's AWS resources. In this case, the provider is an Amazon Cognito user pool, but federating Facebook, Google, SAML and other identity providers is also supported.

Implement Your Sign-in UI

The AWS Mobile SDK provides a library that creates a customizable sign-in UI in your app. To create your sign-in UI, add the following imports and then call the library in the `viewDidLoad()` function of `MasterViewController.swift`.

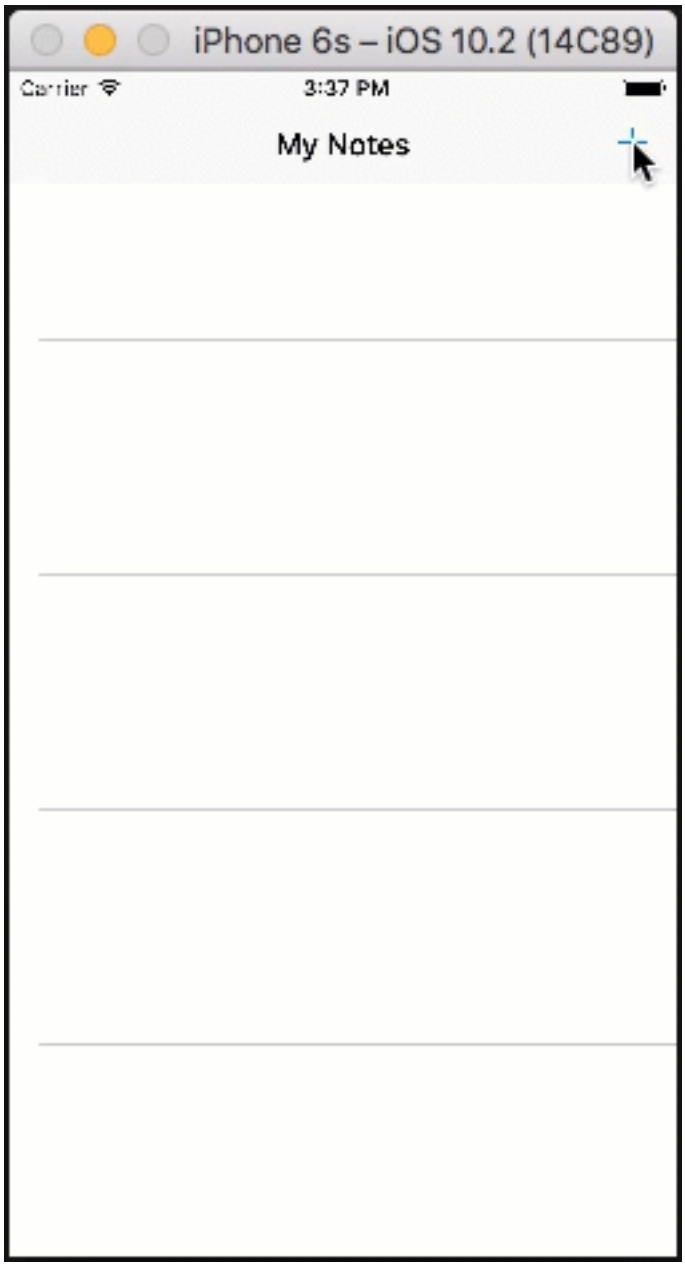
```
import AWSCore
import AWSPinpoint
import UIKit
import AWSAuthCore
import AWSAuthUI

class MasterViewController: UITableViewController, NSFetchedResultsControllerDelegate {

          // . . .

   override func viewDidLoad() {
            super.viewDidLoad()

            // Instantiate sign-in UI from the SDK library
            if !AWSSignInManager.sharedInstance().isLoggedIn {
                AWSAuthUIViewController
                    .presentViewController(with: self.navigationController!,
                         configuration: nil,
                         completionHandler: { (provider: AWSSignInProvider, error: Error?)
 in
                          if error != nil {
                              print("Error occurred: \(String(describing: error))")
                          } else {
                              // Sign in successful.
                          }
                    })
            }

          // . . .
```

```
        }
    }
```

Run the App and Validate Results

Rebuild the project and run in the Simulator. You should see a sign-in screen. Choose the **Create new account** button to create a new account. Once the information is submitted, you will be sent a confirmation code via email. Enter the confirmation code to complete registration, then sign-in with your new account.

Tip	Use Amazon WorkMail as a test email account If you do not want to use your own email account as a test account, create an Amazon WorkMail service within AWS for test accounts. You can get started for free with a 30-day trial for up to 25 accounts.



Next steps

- Continue by integrating NoSQL Data (p. 126).
- Learn more about Amazon Cognito.

Add Online Data Access to the Notes App

In the previous section (p. 121) of this tutorial, we added a simple sign-up / sign-in flow to the sample note-taking app with email validation. This tutorial assumes you have completed the previous tutorials. If you jumped to this step, please go back and start from the beginning (p. 114). In this tutorial, we will add a NoSQL database to our mobile backend, then configure a basic data access service to the note-taking app.

The notes app uses iOS Core Data as a persistence framework. `NotesContentProvider.swift` is custom content provider used as a clean interface for managing your application content locally. In the following steps, you will modify the content provider code to use DynamoDB and sync with the local Core data.

You should be able to complete this section of the tutorial in about 30-45 minutes.

Set Up Your Backend

To add User Sign-in to your app you will create the backend resources in your Mobile Hub project, and then update the configuration file in your app.

Add a NoSQL Database to the AWS Mobile Hub Project

Before we work on the client-side code, we need to add a NoSQL database and table to the backend project:

1. Right-click `awsconfiguration.json` in your Xcode Project Navigator, choose **Delete**, and then choose **Move to trash**.
2. Open the AWS Mobile Hub console.
3. Select your project.
4. Scroll down to the **Add More Backend Features** section and then choose the **NoSQL Database** tile.
5. Choose **Enable NoSQL**, choose **Add Table**, and then choose **Example** to start with an example schema.
6. Choose **Notes**, which most closely matches the model we wish to use.
7. Choose **Add attribute**, then fill in the details of the new attribute:
   - **Attribute name**: `updatedDate`
   - **Type**: `number`
8. Choose **Add index** then fill in the details of the new index:
   - **Index name**: `LastUpdated`
   - **Partition key**: `userId`
   - **Sort key**: `updatedDate`
9. Choose **Create table**
10. Choose **Create table** in the modal dialog.
11. Choose your project name in the upper left and then choose **Integrate** on your iOS app card.
12. Choose **Download Cloud Config** to get an updated `awsconfiguration.json` file.

Connect to Your Backend

To update the linkage between your app and your AWS services:

1. Drag `awsconfiguration.json` from your download location into the Xcode project folder containing `Info.plist`. Select **Copy items if needed** and **Create groups** in the options dialog. Choose **Finish**.

Your system may have modified the filename to avoid conflicts. Make sure the file you add to your Xcode project is named `awsconfiguration.json`.

Download the Models

To aid in implementing a provider for the table you created, Mobile Hub generated a data model descriptor file. To add the data model to your project:

1. Choose your project name in the upper left and then choose **Integrate** on the iOS app card.
2. Choose **Swift Models** under **Download Models**.
3. Unpack the downloaded ZIP file.
4. Find `Notes.swift`, and then drag and drop it into the folder in Xcode that contains file:*Info.plist*. Select **Copy items if needed** and **Create groups** in the options dialog. Choose **Finish**.

Add NoSQL Data Dependencies

1. Add the following NoSQL Data dependencies in your project's `Podfile`

```
platform :ios, '9.0'
target :'MyNotes' do
    use_frameworks!

      # Analytics dependency
      pod 'AWSPinpoint', '~> 2.6.5'

      # Auth dependencies
      pod 'AWSUserPoolsSignIn', '~> 2.6.5'
      pod 'AWSAuthUI', '~> 2.6.5'
      pod 'AWSMobileClient', '~> 2.6.5'

      # NoSQL Data dependencies
      pod 'AWSDynamoDB', '~> 2.6.5'

    # other pods
end
```

Then, in a terminal run:

```
pod install --repo-update
```

If you encounter an error message that begins "`[!] Failed to connect to GitHub to update the CocoaPods/Specs . . .`", and your internet connectivity is working, you may need to update openssl and Ruby.Ï

Implement Mutation Methods

`NotesContentProvider` is the basic interface the app uses to communicate with Core data and your NoSQL table in Amazon DynamoDB. Mutation events handle the CRUD operations when you call its `insertNoteDDB`, `updateNoteDDB`, and `deleteNoteDDB` methods.

To add these mutation methods to the `NotesContentProvider` class, add the following import statement to the file.

```
import AWSDynamoDB
import AWSAuthCore
```

Then add CRUD functions (insert, update, and delete) to the NotesContentProvider to the class as follows.

```
public class NotesContentProvider  {

        // . . .

   //Insert a note using Amazon DynamoDB
```

```
  func insertNoteDDB(noteId: String, noteTitle: String, noteContent: String) -> String {

          let dynamoDbObjectMapper = AWSDynamoDBObjectMapper.default()

          // Create a Note object using data model you downloaded from Mobile Hub
          let noteItem: Notes = Notes()

          noteItem._userId = AWSIdentityManager.default().identityId
          noteItem._noteId = noteId
          noteItem._title = emptyTitle
          noteItem._content = emptyContent
          noteItem._creationDate = NSDate().timeIntervalSince1970 as NSNumber

          //Save a new item
          dynamoDbObjectMapper.save(noteItem, completionHandler: {
              (error: Error?) -> Void in

              if let error = error {
                  print("Amazon DynamoDB Save Error on new note: \(error)")
                  return
              }
              print("New note was saved to DDB.")
          })

          return noteItem._noteId!
      }

  //Insert a note using Amazon DynamoDB
  func updateNoteDDB(noteId: String, noteTitle: String, noteContent: String)  {

          let dynamoDbObjectMapper = AWSDynamoDBObjectMapper.default()

          let noteItem: Notes = Notes()

          noteItem._userId = AWSIdentityManager.default().identityId
          noteItem._noteId = noteId

          if (!noteTitle.isEmpty){
              noteItem._title = noteTitle
          } else {
              noteItem._title = emptyTitle
          }

          if (!noteContent.isEmpty){
              noteItem._content = noteContent
          } else {
              noteItem._content = emptyContent
          }

          noteItem._updatedDate = NSDate().timeIntervalSince1970 as NSNumber
          let updateMapperConfig = AWSDynamoDBObjectMapperConfiguration()
          updateMapperConfig.saveBehavior = .updateSkipNullAttributes //ignore any null
value attributes and does not remove in database
          dynamoDbObjectMapper.save(noteItem, configuration: updateMapperConfig,
completionHandler: {(error: Error?) -> Void in
              if let error = error {
                  print(" Amazon DynamoDB Save Error on note update: \(error)")
                  return
              }
              print("Existing note updated in DDB.")
          })
      }

  //Delete a note using Amazon DynamoDB
  func deleteNoteDDB(noteId: String) {
          let dynamoDbObjectMapper = AWSDynamoDBObjectMapper.default()
```

```
            let itemToDelete = Notes()
            itemToDelete?._userId = AWSIdentityManager.default().identityId
            itemToDelete?._noteId = noteId

            dynamoDbObjectMapper.remove(itemToDelete!, completionHandler: {(error: Error?) -
> Void in
            if let error = error {
                print(" Amazon DynamoDB Save Error: \(error)")
            return
            }
                print("An note was deleted in DDB.")
            })
        }
}
```

Implement Query Methods

This application always asks for the entire data set that the user is entitled to see, so there is no need to implement complex query management. This simplifies the `query()` method considerably. The `query()` method returns a `Cursor` (which is a standard mechanism for iterating over data sets returned from databases).

Add the following query function to the NotesContentProvider class:

```
func getNotesFromDDB() {
        // 1) Configure the query looking for all the notes created by this user (userId =>
 Cognito identityId)
        let queryExpression = AWSDynamoDBQueryExpression()

        queryExpression.keyConditionExpression = "#userId = :userId"

        queryExpression.expressionAttributeNames = [
            "#userId": "userId",
        ]
        queryExpression.expressionAttributeValues = [
            ":userId": AWSIdentityManager.default().identityId
        ]

        // 2) Make the query
        let dynamoDbObjectMapper = AWSDynamoDBObjectMapper.default()

        dynamoDbObjectMapper.query(Notes.self, expression: queryExpression) { (output:
 AWSDynamoDBPaginatedOutput?, error: Error?) in
            if error != nil {
                print("DynamoDB query request failed. Error: \(String(describing: error))")
            }
            if output != nil {
                print("Found [\(output!.items.count)] notes")
                for notes in output!.items {
                    let noteItem = notes as? Notes
                    print("\nNoteId: \(noteItem!._noteId!)\nTitle:
 \(noteItem!._title!)\nContent: \(noteItem!._content!)")
                }
            }
        }
    }
```

Add Data Access Calls

Calls to insert, update, delete, and query data stored in Amazon DynamoDB are made in `MasterViewController` and `DetailsViewController`.

1. To create a note in Amazon DynamoDB , add the following call to `noteContentProvider?.insertNoteDDB()` to the *insert* portion of the `autoSave()` function of `DetailViewController`.

```
// . . .

 // If this is a NEW note, set the Note Id
 if (DetailViewController.noteId == nil) // Insert
 {
     let id = noteContentProvider?.insert(noteTitle: "", noteContent: "")
     noteContentProvider?.insertNoteDDB(noteId: id!, noteTitle: "", noteContent: "")
     DetailViewController.noteId = id
 }
 else // Update
 {
     let noteId = DetailViewController.noteId
     let noteTitle = self.noteTitle.text
     let noteContent = self.noteContent.text
     noteContentProvider?.update(noteId: noteId!, noteTitle: noteTitle!, noteContent:
 noteContent!)
     noteContentProvider?.update(noteId: noteId!, noteTitle: noteTitle!, noteContent:
 noteContent!)
 }

 // . . .
```

2. To update a note from Amazon DynamoDB , add the following line in to the *update* portion of the `autoSave()` function of `DetailViewController`.

```
noteContentProvider?.updateNoteDDB(noteId: noteId!, noteTitle: noteTitle!, noteContent:
 noteContent!)
```

3. To delete a note from Amazon DynamoDB, update the following function in the `MasterViewController` with a call to `deleteNoteDDB()`.

```
override func tableView(_ tableView: UITableView, commit editingStyle:
 UITableViewCellEditingStyle, forRowAt indexPath: IndexPath) {
        if editingStyle == .delete {
            let context = fetchedResultsController.managedObjectContext
            let noteObj = fetchedResultsController.object(at: indexPath)
            let noteId = fetchedResultsController.object(at: indexPath).noteId

            //Delete Note Locally
            _noteContentProvider?.delete(managedObjectContext: context, managedObj:
 noteObj, noteId: noteObj.noteId) //Core Data Delete

            //Delete Note in DynamoDB
            _noteContentProvider?.deleteNoteDDB(noteId: noteId!)
        }
    }
```

4. To query for all notes from Amazon DynamoDB, add the following line to the bottom of the `viewDidLoad()` function in the MasterViewController:

```
_noteContentProvider?.getNotesFromDDB()
```

Note	Differences from a real implementation

	We've taken a simplified approach for this content provider to demonstrate the CRUD implementation. A real implementation would need to deal with online state and handle caching of the data, plus handle appropriate query capabilities as required by the application.

Run the App and Validate Results

You must be online in order to run this application. Run the application in the emulator. Note that the initial startup after logging in is slightly longer (due to reading the data from the remote database).

Data is available immediately in the mobile backend. Create a few notes, then view the records within the AWS Console:

1. Open the Mobile Hub console.
2. Choose your project.
3. Choose **Resources** on the upper right.
4. Choose the link for your Amazon DynamoDB table.
5. Choose the **Items** tab.

When you insert, edit or delete notes in the app, you should be able to see the data on the server reflect your actions almost immediately.

Next Steps

- Learn about Amazon DynamoDB.

AWS Mobile Android and iOS How To

Just Getting Started?	Use streamlined steps (p. 2) to install the SDK and integrate features.

Or, use the contents of this section if your app will integrate existing AWS services.

This section provides information on the steps for achieving specific tasks for integrating your AWS Mobile features into your Android and iOS apps.

Topics

- How To Add Machine Learning with Amazon Machine Learning (p. 294)
- How To For Platform Specific Tasks (p. 300)

How To: AWS Mobile SDK Setup Options

Just Getting Started?	Use streamlined steps (p. 2) to install the SDK and integrate AWS services. *Or, use the contents of this page if your app will integrate existing AWS services.*

This section provides information on how to manually install the AWS Mobile SDK for your Android and iOS apps.

Topics

- Android: Setup Options for the SDK (p. 133)
- iOS: Setup Options for the SDK (p. 138)

Android: Setup Options for the SDK

Just Getting Started?	Use streamlined steps (p. 2) to install the SDK and integrate features.

Or, use the options on this page if your app integrates existing AWS services.

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

Prerequisites

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

- An AWS Account
- Android 2.3.3 (API Level 10) or higher (for more information about the Android platform, see Android Developers)
- Android Studio or Android Development Tools for Eclipse

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

1. Get the AWS Mobile SDK for Android.
2. Set permissions in your `AndroidManifest.xml` file.
3. Obtain AWS credentials using Amazon Cognito.

Step 1: Get the AWS Mobile SDK for Android

There are three ways to get the AWS Mobile SDK for Android.

Option 1: Using Gradle with Android Studio

If you are using Android Studio, add the `aws-android-sdk-core` dependency to your `app/build.gradle` file, along with the dependencies for the individual services that your project will use, as shown below.

```
dependencies {
    implementation 'com.amazonaws:aws-android-sdk-core:2.6.+'
    implementation 'com.amazonaws:aws-android-sdk-s3:2.6.+'
    implementation 'com.amazonaws:aws-android-sdk-ddb:2.6.+'
}
```

A full list of dependencies are listed below. For dependencies ending in "`@aar`" use a compile statement in the following form.

```
implementation ('com.amazonaws:aws-android-sdk-cognitoauth:2.6.+@aar') { transitive =
 true }
```

Dependency	Build.gradle Value
Amazon API Gateway	aws-android-sdk-apigateway-core:2.6.+
AWS Auth Core	aws-android-sdk-auth-core:2.6.+@aar
AWS Facebook SignIn Provider	aws-android-sdk-auth-facebook:2.6.+@aar
AWS Google SignIn Provider	aws-android-sdk-auth-google:2.6.+@aar
AWS Auth UI	aws-android-sdk-auth-ui:2.6.+@aar
AWS Cognito User Pools SignIn Provider	aws-android-sdk-auth-userpools:2.6.+@aar
Amazon Auto Scaling	aws-android-sdk-autoscaling:2.6.+
Amazon CloudWatch	aws-android-sdk-cloudwatch:2.6.+
Amazon Cognito Auth	aws-android-sdk-cognitoauth:2.6.+@aar
Amazon Cognito Identity Provider	aws-android-sdk-cognitoidentityprovider:2.6.+
AWS Core	aws-android-sdk-core:2.6.+
Amazon DynamoDB Document Model	aws-android-sdk-ddb-document:2.6.+
Amazon DynamoDB Object Mapper	aws-android-sdk-ddb-mapper:2.6.+
Amazon DynamoDB	aws-android-sdk-ddb:2.6.+
Amazon Elastic Compute Cloud	aws-android-sdk-ec2:2.6.+
Amazon Elastic Load Balancing	aws-android-sdk-elb:2.6.+
AWS IoT	aws-android-sdk-iot:2.6.+
Amazon Kinesis	aws-android-sdk-kinesis:2.6.+
Amazon Kinesis Video	aws-android-sdk-kinesisvideo:2.6.+@aar
Amazon Key Management Service (KMS)	aws-android-sdk-kms:2.6.+
AWS Lambda	aws-android-sdk-lambda:2.6.+

Dependency	Build.gradle Value
Amazon Lex	aws-android-sdk-lex:2.6.+@aar
Amazon CloudWatch Logs	aws-android-sdk-logs:2.6.+
Amazon Machine Learning	aws-android-sdk-machinelearning:2.6.+
AWS Mobile Client	aws-android-sdk-mobile-client:2.6.+@aar
Amazon Mobile Analytics	aws-android-sdk-mobileanalytics:2.6.+
Amazon Pinpoint	aws-android-sdk-pinpoint:2.6.+
Amazon Polly	aws-android-sdk-polly:2.6.+
Amazon Rekognition	aws-android-sdk-rekognition:2.6.+
Amazon Simple Storage Service (S3)	aws-android-sdk-s3:2.6.+
Amazon Simple DB (SDB)	aws-android-sdk-sdb:2.6.+
Amazon SES	aws-android-sdk-ses:2.6.+
Amazon SNS	aws-android-sdk-sns:2.6.+
Amazon SQS	aws-android-sdk-sqs:2.6.+

Option 2: Import the JAR Files

To obtain the JAR files, download the SDK from http://aws.amazon.com/mobile/sdk. The SDK is stored in a compressed file named `aws-android-sdk-#-#-#`, where #-#-# represents the version number. Source code is available on GitHub.

If using Android Studio:

In the Project view, drag `aws-android-sdk-#-#-#-core.jar` plus the `.jar` files for the individual services your project will use into the `apps/libs` folder. They'll be included on the build path automatically. Then, sync your project with the Gradle file.

If using Eclipse:

Drag the `aws-android-sdk-#-#-#-core.jar` file plus the `.jar` files for the individual services your project will use, into the `libs` folder. They'll be included on the build path automatically.

Option 3: Using Maven

The AWS Mobile SDK for Android supports Apache Maven, a dependency management and build automation tool. A Maven project contains a `pom.xml` file where you can specify the Amazon Web Services that you want to use in your app. Maven then includes the services in your project, so that you don't have to download the entire AWS Mobile SDK and manually include JAR files.

Maven is supported in AWS Mobile SDK for Android v. 2.1.3 and onward. Older versions of the SDK are not available via Maven. If you're new to Maven and you'd like to learn more about it, see the Maven documentation.

pom.xml Example

Here's an example of how you can add Amazon Cognito Identity, Amazon S3, and Amazon Mobile Analytics to your project:

```
<dependencies>
    <dependency>
        <groupid>com.amazonaws</groupid>
        <artifactid>aws-android-sdk-core</artifactid>
        <version>[2.2.0, 2.3)</version>
    </dependency>
    <dependency>
        <groupid>com.amazonaws</groupid>
        <artifactid>aws-android-sdk-s3</artifactid>
        <version>[2.2.0, 2.3)</version>
    </dependency>
    <dependency>
        <groupid>com.amazonaws</groupid>
        <artifactid>aws-android-sdk-mobileanalytics</artifactid>
        <version>[2.2.0, 2.3)</version>
    </dependency>
</dependencies>
```

As shown above, the groupId for the AWS Mobile SDK for Android is `com.amazonaws`. For each additional service, include a `<dependency>` element following the model above, and use the appropriate artifactID from the table below. The `<version>` element specifies the version of the AWS Mobile SDK for Android. The example above demonstrate's Maven's ability to use a range of acceptable versions for a given dependency. To review available versions of the SDK for Android, see the Release Notes.

The AWS Mobile `artifactId` values are as follows:

Service/Feature	artifactID
Amazon API Gateway	aws-android-sdk-apigateway-core
AWS Auth Core	aws-android-sdk-auth-core
AWS Facebook SignIn Provider	aws-android-sdk-auth-facebook
AWS Google SignIn Provider	aws-android-sdk-auth-google
AWS Auth UI	aws-android-sdk-auth-ui
AWS Cognito User Pools SignIn Provider	aws-android-sdk-auth-userpools
Amazon Auto Scaling	aws-android-sdk-autoscaling
Amazon CloudWatch	aws-android-sdk-cloudwatch
Amazon Cognito Auth	aws-android-sdk-cognitoauth
Amazon Cognito Identity Provider	aws-android-sdk-cognitoidentityprovider
AWS Core	aws-android-sdk-core
Amazon DynamoDB Document Model	aws-android-sdk-ddb-document
Amazon DynamoDB Object Mapper	aws-android-sdk-ddb-mapper
Amazon DynamoDB	aws-android-sdk-ddb
Amazon Elastic Compute Cloud	aws-android-sdk-ec2
Amazon Elastic Load Balancing	aws-android-sdk-elb
AWS IoT	aws-android-sdk-iot

Service/Feature	artifactID
Amazon Kinesis	aws-android-sdk-kinesis
Amazon Kinesis Video	aws-android-sdk-kinesisvideo
Amazon Key Management Service (KMS)	aws-android-sdk-kms
AWS Lambda	aws-android-sdk-lambda
Amzon Lex	aws-android-sdk-lex
Amazon CloudWatch Logs	aws-android-sdk-logs
Amazon Machine Learning	aws-android-sdk-machinelearning
AWS Mobile Client	aws-android-sdk-mobile-client
Amazon Mobile Analytics	aws-android-sdk-mobileanalytics
Amazon Pinpoint	aws-android-sdk-pinpoint
Amazon Polly	aws-android-sdk-polly
Amazon Rekognition	aws-android-sdk-rekognition
Amazon Simple Storage Service (S3)	aws-android-sdk-s3
Amazon Simple DB (SDB)	aws-android-sdk-sdb
Amazon SES	aws-android-sdk-ses
Amazon SNS	aws-android-sdk-sns
Amazon SQS	aws-android-sdk-sqs

Step 2: Set Permissions in Your Manifest

Add the following permission to your `AndroidManifest.xml`

```
<uses-permission android:name="android.permission.INTERNET" />
```

Step 3: Get AWS Credentials

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

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

To create an identity pool for your application:

1. Log in to the Amazon Cognito Console and click **Manage Federated Identities**, then **Create new identity pool**.
2. Enter a name for your Identity Pool and check the checkbox to enable access to unauthenticated identities. Click **Create Pool** to create your identity pool.

3. Click **Allow** to create the two default roles associated with your identity pool—one for unauthenticated users and one for authenticated users.

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

```
CognitoCachingCredentialsProvider credentialsProvider = new
 CognitoCachingCredentialsProvider(
    getApplicationContext(),    /* get the context for the application */
    "COGNITO_IDENTITY_POOL",    /* Identity Pool ID */
    Regions.MY_REGION           /* Region for your identity pool--US_EAST_1 or EU_WEST_1*/
);
```

Next Steps

- **Run the demos**: View our sample Android apps that demonstrate common use cases. To run the sample apps, set up the SDK for Android as described above, and then follow the instructions contained in the README files of the individual samples.
- **Read the API Reference**: View the API Reference for the AWS Mobile SDK for Android.
- **Try AWS Mobile Hub**: Quickly configure and provision an AWS cloud backend for many common mobile app features, and download end to end working Android demonstration projects, SDK, and helper code, all generated based on your choices.
- **Ask questions**: Post questions on the AWS Mobile SDK Forums.

iOS: Setup Options for the SDK

Just Getting Started?	Use streamlined steps (p. 2) to install the SDK and integrate AWS features.

Or, use the contents of this page if your app will integrate existing AWS services.

Topics

To add the SDK, install the following on your development machine:

- Xcode 7 or later
- iOS 8 or later

You can view the source code in the AWS Mobile SDK for iOS GitHub repository.

Include the AWS Mobile SDK for iOS in an Existing Application

The samples included with the SDK are standalone projects that are already set up. You can also integrate the SDK into your own existing project. Choose one of the following three ways to import the SDK into your project:

- Cocoapods
- Carthage
- Dynamic Frameworks

Note
Importing the SDK in multiple ways loads duplicate copies of the SDK into the project and causes compiler errors.

CocoaPods

1. The AWS Mobile SDK for iOS is available through CocoaPods. Install CocoaPods by running the following commands from the folder containing your projects `*.xcodeproj` file.

   ```
   $ gem install cocoapods
   ```

   Note
   Depending on your system settings, you may need to run the command as administrator using *sudo*, as follows:

   ```
   $ sudo gem install cocoapods
   ```

   ```
   $ pod setup
   ```

   ```
   $ pod init
   ```

2. In your project directory (the directory where your `*.xcodeproj` file is), open the empty text file named `Podfile` (without a file extension) and add the following lines to the file. Replace `myAppName` with your app name. You can also remove pods for services that you don't use. For example, if you don't use *AWSAutoScaling*, remove or do not include the `AWSAutoScaling` pod.

   ```
   source 'https://github.com/CocoaPods/Specs.git'

   platform :ios, '8.0'
   use_frameworks!

   target :'YOUR-APP-NAME' do
       pod 'AWSAuth'
       pod 'AWSAuthCore'
       pod 'AWSAuthUI'
       pod 'AWSAutoScaling'
       pod 'AWSCloudWatch'
       pod 'AWSCognito'
       pod 'AWSCognitoAuth'
       pod 'AWSCognitoIdentityProvider'
       pod 'AWSCognitoIdentityProviderASF'
       pod 'AWSCore'
       pod 'AWSDynamoDB'
       pod 'AWSEC2'
       pod 'AWSElasticLoadBalancing'
       pod 'AWSFacebookSignIn'
       pod 'AWSGoogleSignIn'
       pod 'AWSIoT'
       pod 'AWSKMS'
       pod 'AWSKinesis'
       pod 'AWSLambda'
       pod 'AWSLex'
       pod 'AWSLogs'
       pod 'AWSMachineLearning'
       pod 'AWSMobileAnalytics'
       pod 'AWSMobileClient'
       pod 'AWSPinpoint'
       pod 'AWSPolly'
   ```

```
    pod 'AWSRekognition'
    pod 'AWSS3'
    pod 'AWSSES'
    pod 'AWSSNS'
    pod 'AWSSQS'
    pod 'AWSSimpleDB'
    pod 'AWSUserPoolsSignIn'
end
```

3. Run the following command:

   ```
   $ pod install
   ```

4. Open `*.xcworkspace` with Xcode, rebuild your app, and start using the SDK.

   Note
   Once you have created a workspace, always use **.xcworkspace* to open the project instead of `*.xcodeproj`.

5. Rebuild your app after reopening it in the workspace to resolve APIs from new libraries called in your code. This is a good practice any time you add import statements.

Carthage

1. Install the latest version of Carthage.
2. Add the following to your *Cartfile*:

   ```
   github "aws/aws-sdk-ios"
   ```

3. Run the following command:

   ```
   $ carthage update
   ```

4. With your project open in Xcode, choose your **Target**. In the **General** tab, find **Embedded Binaries**, then choose the **+** button.
5. Choose the **Add Other** button, navigate to the `AWS<#ServiceName#>.framework` files under **Carthage** > **Build** > **iOS** and select `AWSCore.framework` and the other service frameworks you require. Do not select the **Destination: Copy items if needed** checkbox when prompted.
   - `AWSAuth`
   - `AWSAuthCore`
   - `AWSAuthUI`
   - `AWSAutoScaling`
   - `AWSCloudWatch`
   - `AWSCognito`
   - `AWSCognitoAuth`
   - `AWSCognitoIdentityProvider`
   - `AWSCognitoIdentityProviderASF`
   - `AWSCore`
   - `AWSDynamoDB`
   - `AWSEC2`
   - `AWSElasticLoadBalancing`
   - `AWSFacebookSignIn`
   - `AWSGoogleSignIn`
   - `AWSIoT`
   - `AWSKMS`
   - `AWSKinesis`

- `AWSLambda`
- `AWSLex`
- `AWSLogs`
- `AWSMachineLearning`
- `AWSMobileAnalytics`
- `AWSMobileClient`
- `AWSPinpoint`
- `AWSPolly`
- `AWSRekognition`
- `AWSS3`
- `AWSSES`
- `AWSSNS`
- `AWSSQS`
- `AWSSimpleDB`
- `AWSUserPoolsSignIn`

6. Under the **Build Phases** tab in your **Target**, choose the **+** button on the top left and then select **New Run Script Phase**.

Setup the build phase as follows. Make sure this phase is below the **Embed Frameworks** phase.

```
Shell /bin/sh

bash "${BUILT_PRODUCTS_DIR}/${FRAMEWORKS_FOLDER_PATH}/AWSCore.framework/strip-
frameworks.sh"

Show environment variables in build log: Checked
Run script only when installing: Not checked

Input Files: Empty
Output Files: Empty
```

Frameworks

1. Download the SDK from http://aws.amazon.com/mobile/sdk. The SDK is stored in a compressed file archive named `aws-ios-sdk-#.#.#`, where '#.#.#' represents the version number. For version 2.5.0, the filename is `aws-ios-sdk-2.5.0`.
2. With your project open in Xcode, choose your **Target**. Under the **General** tab, find **Embedded Binaries** and then choose the **+** button.
3. Choose **Add Other**. Navigate to the `AWS<#ServiceName#>.framework` files and select `AWSCore.framework` and the other service frameworks you require. Select the **Destination: Copy items if needed** checkbox when prompted.
   - `AWSAuth`
   - `AWSAuthCore`
   - `AWSAuthUI`
   - `AWSAutoScaling`
   - `AWSCloudWatch`
   - `AWSCognito`
   - `AWSCognitoAuth`
   - `AWSCognitoIdentityProvider`
   - `AWSCognitoIdentityProviderASF`

- `AWSCore`
- `AWSDynamoDB`
- `AWSEC2`
- `AWSElasticLoadBalancing`
- `AWSFacebookSignIn`
- `AWSGoogleSignIn`
- `AWSIoT`
- `AWSKMS`
- `AWSKinesis`
- `AWSLambda`
- `AWSLex`
- `AWSLogs`
- `AWSMachineLearning`
- `AWSMobileAnalytics`
- `AWSMobileClient`
- `AWSPinpoint`
- `AWSPolly`
- `AWSRekognition`
- `AWSS3`
- `AWSSES`
- `AWSSNS`
- `AWSSQS`
- `AWSSimpleDB`
- `AWSUserPoolsSignIn`

4. Under the **Build Phases** tab in your **Target**, click the **+** button on the top left and then select **New Run Script Phase**.
5. Setup the build phase as follows. Make sure this phase is below the *Embed Frameworks* phase.

```
Shell /bin/sh

bash "${BUILT_PRODUCTS_DIR}/${FRAMEWORKS_FOLDER_PATH}/AWSCore.framework/strip-
frameworks.sh"

Show environment variables in build log: Checked
Run script only when installing: Not checked

Input Files: Empty
Output Files: Empty
```

Update the SDK to a Newer Version

This section describes how to pick up changes when a new SDK is released.

CocoaPods

Run the following command in your project directory. CocoaPods automatically picks up the changes.

```
$ pod update
```

Note
If your pod update command fails, delete `Podfile.lock` and `Pods/` and then run **`pod install`** to cleanly install the SDK.

Carthage

Run the following command in your project directory. Carthage automatically updates your frameworks with the new changes.

```
$ carthage update
```

Frameworks

1. In Xcode select the following frameworks in **Project Navigator** and press the **delete** key. Then select **Move to Trash**:
   - `AWSAuth`
   - `AWSAuthCore`
   - `AWSAuthUI`
   - `AWSAutoScaling`
   - `AWSCloudWatch`
   - `AWSCognito`
   - `AWSCognitoAuth`
   - `AWSCognitoIdentityProvider`
   - `AWSCognitoIdentityProviderASF`
   - `AWSCore`
   - `AWSDynamoDB`
   - `AWSEC2`
   - `AWSElasticLoadBalancing`
   - `AWSFacebookSignIn`
   - `AWSGoogleSignIn`
   - `AWSIoT`
   - `AWSKMS`
   - `AWSKinesis`
   - `AWSLambda`
   - `AWSLex`
   - `AWSLogs`
   - `AWSMachineLearning`
   - `AWSMobileAnalytics`
   - `AWSMobileClient`
   - `AWSPinpoint`
   - `AWSPolly`
   - `AWSRekognition`
   - `AWSS3`
   - `AWSSES`
   - `AWSSNS`
   - `AWSSQS`
   - `AWSSimpleDB`
   - `AWSUserPoolsSignIn`
2. Follow the Frameworks installation steps in the previous section, to include the new version of the SDK.

Logging

As of version 2.5.4 of this SDK, logging utilizes CocoaLumberjack SDK, a flexible, fast, open source logging framework. It supports many capabilities including the ability to set logging level per output target, for instance, concise messages logged to the console and verbose messages to a log file.

CocoaLumberjack logging levels are additive such that when the level is set to verbose, all messages from the levels below verbose are logged. It is also possible to set custom logging to meet your needs. For more information, see CocoaLumberjack Logging Levels

Changing Logging Level

You can change the logging level to suit the phase of your development cycle by importing AWSCore and calling:

iOS - Swift

```
AWSDDLog.sharedInstance().logLevel = .verbose
```

The following logging level options are available:

- `.off`
- `.error`
- `.warning`
- `.info`
- `.debug`
- `.verbose`

We recommend setting the log level to `.off` before publishing to the App Store.

iOS - Objective-C

```
[AWSDDLog sharedInstance].logLevel = AWSDDLogLevelVerbose;
```

The following logging level options are available:

- `AWSDDLogLevelOff`
- `AWSDDLogLevelError`
- `AWSDDLogLevelWarning`
- `AWSDDLogLevelInfo`
- `AWSDDLogLevelDebug`
- `AWSDDLogLevelVerbose`

We recommend setting the log level to `AWSDDLogLevelOff` before publishing to the App Store.

Targeting Log Output

CocoaLumberjack can direct logs to file or used as a framework that integrates with the Xcode console.

To initialize logging to files, use the following code:

iOS - Swift

```
let fileLogger: AWSDDFileLogger = AWSDDFileLogger() // File Logger
```

```
fileLogger.rollingFrequency = TimeInterval(60*60*24)  // 24 hours
fileLogger.logFileManager.maximumNumberOfLogFiles = 7
AWSDDLog.add(fileLogger)
```

iOS - Objective-C

```
AWSDDFileLogger *fileLogger = [[AWSDDFileLogger alloc] init]; // File Logger
fileLogger.rollingFrequency = 60 * 60 * 24; // 24 hour rolling
fileLogger.logFileManager.maximumNumberOfLogFiles = 7;
[AWSDDLog addLogger:fileLogger];
```

To initialize logging to your Xcode console, use the following code:

iOS - Swift

```
AWSDDLog.add(AWSDDTTYLogger.sharedInstance) // TTY = Xcode console
```

iOS - Objective-C

```
[AWSDDLog addLogger:[AWSDDTTYLogger sharedInstance]]; // TTY = Xcode console
```

To learn more, see CocoaLumberjack on GitHub.

Sample Apps

The AWS Mobile SDK for iOS includes sample apps that demonstrate common use cases.

Amazon Cognito Your User Pools Sample (Objective-C)

This sample demonstrates how sign up and sign in a user to display an authenticated portion of your app.

AWS services demonstrated:

- Amazon Cognito Pools
- Amazon Cognito Identity

Amazon DynamoDB Object Mapper Sample (Swift, Objective-C)

This sample demonstrates how to insert, update, delete, query items using DynamoDBObjectMapper.

AWS services demonstrated:

- Amazon DynamoDB
- Amazon Cognito Identity

Amazon S3 Transfer Utility Sample (Swift, Objective-C)

This sample demonstrates how to use the Amazon S3 TransferUtility to download / upload files.

AWS services demonstrated:

- Amazon S3
- Amazon Cognito Identity

Install the Reference Documentation in Xcode

The AWS Mobile SDK for iOS includes documentation in the DocSets format that you can view within Xcode. The easiest way to install the documentation is to use the macOS terminal.

To install the DocSet for Xcode

Open the macOS terminal and go to the directory containing the expanded archive. For example:

```
$ cd ~/Downloads/aws-ios-sdk-2.5.0
```

> **Note**
> Replace `2.5.0` in the preceding example with the version number of the AWS Mobile SDK for iOS that you downloaded.

Create a directory called `~/Library/Developer/Shared/Documentation/DocSets`:

```
$ mkdir -p ~/Library/Developer/Shared/Documentation/DocSets
```

Copy (or move) `documentation/com.amazon.aws.ios.docset` from the SDK installation files to the directory you created in the previous step:

```
$ mv documentation/com.amazon.aws.ios.docset ~/Library/Developer/Shared/
Documentation/DocSets/
```

If Xcode was running during this procedure, restart Xcode. To browse the documentation, go to **Help**, click **Documentation and API Reference**, and select **AWS Mobile SDK for iOS v2.0 Documentation** (where '2.0' is the appropriate version number).

Next Steps

- **Run the demos**: View our sample iOS apps that demonstrate common use cases. To run the sample apps, set up the SDK for iOS as described above, and then follow the instructions contained in the README files of the individual samples.
- **Read the API Reference**: View the API Reference for the AWS Mobile SDK for Android.
- **Try AWS Mobile Hub**: Quickly configure and provision an AWS cloud backend for many common mobile app features, and download end to end working iOS demonstration projects, SDK, and helper code, all generated based on your choices.
- **Ask questions**: Post questions on the AWS Mobile SDK Forums.

How To: User Sign-in with Amazon Cognito

Just Getting Started?	Use streamlined steps (p. 20) to install the SDK and integrate Amazon Cognito. *Or, use the contents of this page if your app will integrate existing AWS services.

This section provides information on the steps for achieving specific tasks for integrating User SignIn features into your Android and iOS apps.

Topics

- Set Up Google Authentication (p. 161)
- Setting Up Custom Authentication (p. 179)
- Customize the SDK Sign-In UI (p. 179)

How to Integrate Your Existing Identity Pool

Just Getting Started?	Use streamlined steps (p. 20) to install the SDK and integrate Amazon Cognito.

The Get Started (p. 20) section of this guide allows you to create new resources and complete the steps described on this page in minutes. If you want to import existing resources or create them from scratch, this page will walk you through the following steps:

- Set up short-lived credentials for accessing your AWS resources using a Cognito Identity Pool.
- Create an AWS Mobile configuration file that ties your app code to the identity pool that enables users to acces your AWS resources.
- Make small adjustments to your app code to install the SDK and retrieve AWS credentials for your user.

Set Up Your Backend

Import or Create a New Identity Pool

- If you already have an Amazon Cognito Identity Pool and know its ID and region, you can skip to Connect to Your Backend (p. 147).

To create a new identity pool:

1. Go to Amazon Cognito Console and choose **Manage Federated Identities**.
2. Choose **Create new Identity pool** on the top left of the console.
3. Type a name for the Identity pool, select **Enable access to unauthenticated identities** under the **Unauthenticated Identities** section, and then choose **Create pool** on the bottom right.
4. Expand the **View Details** section to see the two roles that are to be created to enable access to your bucket. Copy and keep both the Authenticated and Unauthenticated role names, in the form of `Cognito_<IdentityPoolName>Auth_Role` and `Cognito_<IdentityPoolName>Unauth_Role`. In many cases, you will modify the permissions policy of these roles to control access to AWS resources that you add to your app.
5. Choose **Allow** on the bottom right.
6. In the code snippet labeled **Get AWSCredentials** displayed by the console, copy the Identity Pool ID and the Region for use in a following configuration step. You will use these values to connect your backend to your app.

Connect to Your Backend

Take the following steps to connect your app to its backedn.

Topics

- Create the awsconfiguration.json file (p. 148)
- Add the awsconfiguration.json file to your app (p. 148)
- Add the SDK to your App (p. 149)

Create the awsconfiguration.json file

1. Create a file with name `awsconfiguration.json` with the following contents:

```
{
    "Version": "1.0",
    "CredentialsProvider": {
        "CognitoIdentity": {
            "Default": {
                "PoolId": "COGNITO-IDENTITY-POOL-ID",
                "Region": "COGNITO-IDENTITY-POOL-REGION"
            }
        }
    },
    "IdentityManager" : {
      "Default" : {

      }
    }
}
```

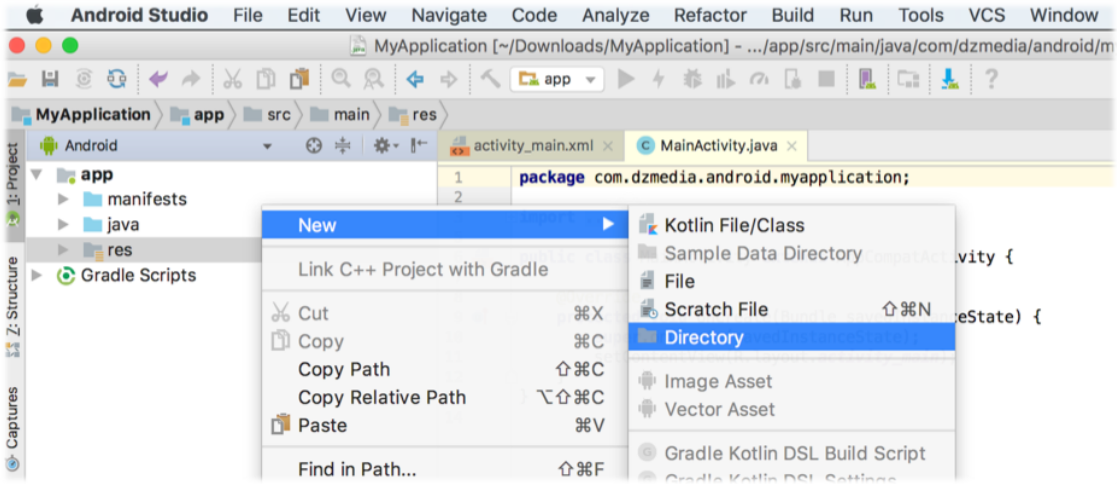
2. Make the following changes to the configuration file.
   - Replace the `COGNITO-IDENTITY-POOL-ID` with the identity pool ID.
   - Replace the `COGNITO-IDENTITY-POOL-REGION` with the region the identity pool was created in.

Need to find your pool's ID and region?	Go to Amazon Cognito Console and choose **Manage Federated Identities**, then choose your pool and choose **Edit identity pool**. Copy the value of **Identity pool ID**.
	
	Insert this region value into the following form to create the value you need for this integration.
	
	`"Region": "REGION-PREFIX-OF-YOUR-POOL-ID".`
	
	For example, if your pool ID is `us-east-1:01234567-yyyy-0123-xxxx-012345678901`, then your integration region value would be:
	
	`"Region": "us-east-1"`

Add the awsconfiguration.json file to your app

Android - Java

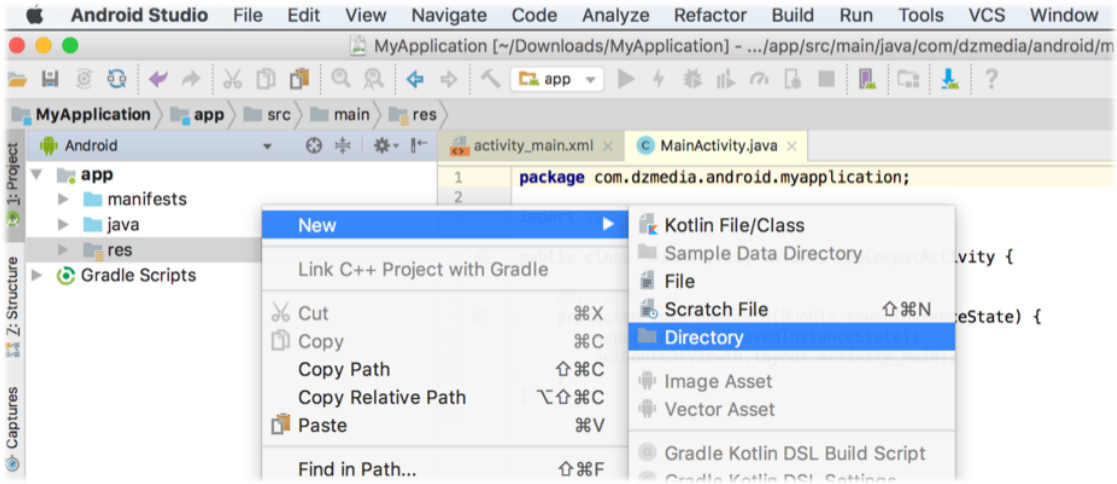
In the Android Studio Project Navigator, right-click your app's `res` folder, and then choose **New > Directory**. Type `raw` as the directory name and then choose **OK**.



Drag the `awsconfiguration.json` you created into the `res/raw` folder. Android gives a resource ID to any arbitrary file placed in this folder, making it easy to reference in the app.

Android - Kotlin

In the Android Studio Project Navigator, right-click your app's `res` folder, and then choose **New > Directory**. Type `raw` as the directory name and then choose **OK**.



Drag the `awsconfiguration.json` you created into the `res/raw` folder. Android gives a resource ID to any arbitrary file placed in this folder, making it easy to reference in the app.

iOS - Swift

Drag the `awsconfiguration.json` into the folder containing your `Info.plist` file in your Xcode project. Choose **Copy items** and **Create groups** in the options dialog.

Add the SDK to your App

Android - Java

Set up AWS Mobile SDK components as follows:

1. Add the following to `app/build.gradle`:

```
dependencies {
```

```
   implementation ('com.amazonaws:aws-android-sdk-mobile-client:2.6.+@aar')
 { transitive = true }

   // other dependencies . . .
}
```

2. Perform a Gradle sync to download the AWS Mobile SDK components into your app.
3. Add the following code to the `onCreate` method of your main or startup activity. This will establish a connection with AWS Mobile. `AWSMobileClient` is a singleton that will be an interface for your AWS services.

   Once the network call to retrieve the user's AWS identity ID has succeeded, you can get the users identity using `getCachedUserID()` from the `AWSIdentityManager`.

```
import com.amazonaws.auth.AWSCredentialsProvider;
import com.amazonaws.mobile.auth.core.IdentityHandler;
import com.amazonaws.mobile.auth.core.IdentityManager;
import com.amazonaws.mobile.client.AWSMobileClient;
import com.amazonaws.mobile.client.AWSStartupHandler;
import com.amazonaws.mobile.client.AWSStartupResult;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        AWSMobileClient.getInstance().initialize(this, new AWSStartupHandler() {
            @Override
            public void onComplete(AWSStartupResult awsStartupResult) {

                //Make a network call to retrieve the identity ID
                // using IdentityManager. onIdentityId happens UPon success.
                IdentityManager.getDefaultIdentityManager().getUserID(new
 IdentityHandler() {

                    @Override
                    public void onIdentityId(String s) {

                        //The network call to fetch AWS credentials succeeded, the
 cached
                        // user ID is available from IdentityManager throughout your
 app
                        Log.d("MainActivity", "Identity ID is: " + s);
                        Log.d("MainActivity", "Cached Identity ID: " +
 IdentityManager.getDefaultIdentityManager().getCachedUserID());
                    }

                    @Override
                    public void handleError(Exception e) {
                        Log.e("MainActivity", "Error in retrieving Identity ID: " +
 e.getMessage());
                    }
                });
            }
        }).execute();
    }
}
```

When you run your app, you should see no behavior change. To verify success, look for the message `"Welcome to AWS!"` in your debug output.

Android - Kotlin

Set up AWS Mobile SDK components as follows:

1. Add the following to `app/build.gradle`:

```
dependencies {
   implementation ('com.amazonaws:aws-android-sdk-mobile-client:2.6.+@aar')
 { transitive = true }

   // other dependencies . . .
}
```

2. Perform a Gradle sync to download the AWS Mobile SDK components into your app.
3. Add the following code to the `onCreate` method of your main or startup activity. This will establish a connection with AWS Mobile. `AWSMobileClient` is a singleton that will be an interface for your AWS services.

   Once the network call to retrieve the user's AWS identity ID has succeeded, you can get the users identity using `getCachedUserID()` from the `AWSIdentityManager`.

```
import com.amazonaws.auth.AWSCredentialsProvider;
import com.amazonaws.mobile.auth.core.IdentityHandler;
import com.amazonaws.mobile.auth.core.IdentityManager;
import com.amazonaws.mobile.client.AWSMobileClient;
import com.amazonaws.mobile.client.AWSStartupHandler;
import com.amazonaws.mobile.client.AWSStartupResult;

class MainActivity : AppCompatActivity() {
    override fun onCrearte(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        AWSMobileClient.getInstance().initialize(this) {
            IdentityManager.defaultIdentityManager.getUserID(
                object : IdentityHandler() {
                    override fun onIdentityId(s: String) {
                        // The netwirk call to fetch AWS credentials succeeded
                        Log.d(TAG, "Identity ID is: ${s}")
                    }

                    override fun handleError(ex: Exception) {
                        Log.e(TAG, "Error: ${ex.message}")
                    }
                }
            )
        }.execute()
    }
}
```

   When you run your app, you should see no behavior change. To verify success, look for the message `"Welcome to AWS!"` in your debug output.

iOS - Swift

Set up AWS Mobile SDK components as follows:

1. Add the `AWSMobileClient` pod to your `Podfile` to install the AWS Mobile SDK.

```
platform :ios, '9.0'
```

```
  target :'YOUR-APP-NAME' do
     use_frameworks!

      pod 'AWSMobileClient', '~> 2.6.13'

      # other pods . . .

  end
```

2. Run `pod install --repo-update` in your app root folder before you continue.

   If you encounter an error message that begins "`[!] Failed to connect to GitHub to update the CocoaPods/Specs . . .`", and your internet connectivity is working, you may need to update openssl and Ruby.

3. Add the following code to your AppDelegate to establish a run-time connection with AWS Mobile.

```
import UIKit
import AWSMobileClient

@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate {

  func application(_ application: UIApplication,
        didFinishLaunchingWithOptions launchOptions:

        [UIApplicationLaunchOptionsKey: Any]?) -> Bool {


        // Uncomment to turn on logging, look for "Welcome to AWS!" to confirm
 success
        // AWSDDLog.add(AWSDDTTYLogger.sharedInstance)
        // AWSDDLog.sharedInstance.logLevel = .info


        // Instantiate AWSMobileClient to get AWS user credentials
        return AWSMobileClient.sharedInstance().interceptApplication(application,
 didFinishLaunchingWithOptions: launchOptions)

  }
}
```

   When you run your app, you should see no behavior change. To verify success, turn on logging by uncommenting the lines in the preceding example, and look for the message "`Welcome to AWS!`" in your the output.

4. To get the users identity, use `getCredentialsProvider()` to access `AWSIdentityManager`, shown here being done in a `ViewController`.

```
import UIKit
import AWSMobileClient
import AWSAuthCore

class ViewController: UIViewController {

    @IBOutlet weak var textfield: UITextField!
    override func viewDidLoad() {
        super.viewDidLoad()
        textfield.text = "View Controller Loaded"

        // Get the identity Id from the AWSIdentityManager
        let appDelegate = UIApplication.shared.delegate as! AppDelegate
        let credentialsProvider =
 AWSMobileClient.sharedInstance().getCredentialsProvider()
```

```
        let identityId = AWSIdentityManager.default().identityId
    }
}
```

Next Steps

- For further information, see Amazon Cognito Developer Guide.

Sign-out a Signed-in User

Prerequisite	This section describes how to add sign-out flow for a signed-in user. In the following example, the built-in AWS Mobile SDK sign-in/sign-up screen is displayed after the user signs out. The examples on this page assume that you have added the AWS Mobile SDK to your mobile app using the steps on the Add User Sign-in page, and have configured an identity provider.

Enable User Sign-out

Android - Java

In the following example, `AWSMobileClient` is instantiated within the `onCreate` method of an activity called `AuthenticatorActivity`. If the client does not find a cached identity from a previous sign-in, it retrieves an unauthenticated "guest" Amazon Cognito Federated Identity ID that is used to access other AWS services. In Logcat, look for the string: `Welcome to AWS!` to see that the client has successfully instantiated.

If the user already has a cached authenticated identity ID from a previous sign-in, then `AWSMobileClient` will resume the session without an additional sign-in.

A `SignInStateChangeListener` object is added to `IdentityManager`, which captures `onUserSignedIn` and `onUserSignedOut` events.

Finally, `showSignIn()` is called to create a `SignInUI` object, and to call the object's `login` method. This displays the built-in sign-in UI of the SDK, and defines `MainActivity` as the navigation target of a successful sign-in. The `SignInUI` calls are placed in a separate function so they can also easily be called when the `onUserSignedOut` event fires.

In Logcat, a successful sign-in prints the string; `Sign-in succeeded`.

```
// AuthenticatorActivity.java

package com.YOUR-DOMAIN.android.YOUR-APP-NAME;

import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
import android.widget.TextView;

// AWSMobileClient imports
import com.amazonaws.mobile.client.AWSMobileClient;
import com.amazonaws.mobile.client.AWSStartupHandler;
```

```
import com.amazonaws.mobile.client.AWSStartupResult;

// AWS SDK sign-in UI imports
import com.amazonaws.mobile.auth.core.IdentityHandler;
import com.amazonaws.mobile.auth.core.IdentityManager;
import com.amazonaws.mobile.auth.core.SignInStateChangeListener;
import com.amazonaws.mobile.auth.ui.SignInUI;

public class AuthenticatorActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_authenticator);

        AWSMobileClient.getInstance().initialize(this).execute();

        // Sign-in listener
        IdentityManager.getDefaultIdentityManager().addSignInStateChangeListener(new
 SignInStateChangeListener() {
            @Override
            public void onUserSignedIn() {
                Log.d(LOG_TAG, "User Signed In");
            }

            // Sign-out listener
            @Override
            public void onUserSignedOut() {

                Log.d(LOG_TAG, "User Signed Out");
                showSignIn();
            }
        });

        showSignIn();
    }


    /*
     * Display the AWS SDK sign-in/sign-up UI
     */
    private void showSignIn() {

        Log.d(LOG_TAG, "showSignIn");

        SignInUI signin = (SignInUI)
 AWSMobileClient.getInstance().getClient(AuthenticatorActivity.this, SignInUI.class);
        signin.login(AuthenticatorActivity.this, MainActivity.class).execute();
    }
}
```

`MainActivity` displays a sign-out button, that calls the `signOut()` method of the `IdentityManager`. This will fire the `SignInStateChangeListener.onSignedOut()` event defined in the `AuthenticatorActivity`. In Logcat, you should see the string: `Signing out...`.

`onUserSignedOut()` then calls `showSignIn` which causes the sign-in screen to reappear.

```
package com.dzmedia.android.YOUR-APP-NAME;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.Button;
```

```
import android.widget.TextView;


import com.amazonaws.mobile.auth.core.IdentityHandler;
import com.amazonaws.mobile.auth.core.IdentityManager;
import com.amazonaws.mobile.client.AWSMobileClient;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

            // Create log out Button on click listener
            Button clickButton = (Button) findViewById(R.id.signOutButton);
            clickButton.setOnClickListener( new View.OnClickListener() {

              public void onClick(View v) {
                  IdentityManager.getDefaultIdentityManager().signOut();
              }
        });
    }
    // other MainActivity code . . .
}
```

Android - Kotlin

In the following example, `AWSMobileClient` is instantiated within the `onCreate` method of an activity called `AuthenticatorActivity`. If the client does not find a cached identity from a previous sign-in, it retrieves an unauthenticated "guest" Amazon Cognito Federated Identity ID that is used to access other AWS services. In Logcat, look for the string: `Welcome to AWS!` to see that the client has successfully instantiated.

If the user already has a cached authenticated identity ID from a previous sign-in, then `AWSMobileClient` will resume the session without an additional sign-in.

A `SignInStateChangeListener` object is added to `IdentityManager`, which captures `onUserSignedIn` and `onUserSignedOut` events.

Finally, `showSignIn()` is called to create a `SignInUI` object, and to call the object's `login` method. This displays the built-in sign-in UI of the SDK, and defines `MainActivity` as the navigation target of a successful sign-in. The `SignInUI` calls are placed in a separate function so they can also easily be called when the `onUserSignedOut` event fires.

In Logcat, a successful sign-in prints the string; `Sign-in succeeded`.

```
// AuthenticatorActivity.java

package com.your-domain.android.YOUR-APP-NAME;

import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
import android.widget.TextView;

// AWSMobileClient imports
import com.amazonaws.mobile.client.AWSMobileClient;
import com.amazonaws.mobile.client.AWSStartupHandler;
import com.amazonaws.mobile.client.AWSStartupResult;

// AWS SDK sign-in UI imports
import com.amazonaws.mobile.auth.core.IdentityHandler;
```

```
import com.amazonaws.mobile.auth.core.IdentityManager;
import com.amazonaws.mobile.auth.core.SignInStateChangeListener;
import com.amazonaws.mobile.auth.ui.SignInUI;

class AuthenticatorActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        AWSMobileClient.getInstance().initialize(this).execute()

        // Sign-in listener
        IdentityManager.defaultIdentityManager.addSignInStateChangeListener(
            object : SignInStateChangeListener() {
                override fun onUserSignedIn() {
                    // Do something
                }

                override fun onUserSignedOut() {
                    showSignIn()
                }
            }
        )
        showSignIn()
    }

    private fun showSignIn() {
        val ui = AWSMobileClient.getInstance().getClient(this@AuthenticatorActivity,
 SignInUI::class.java)
        ui.login(this@AuthenticatorActivity, MainActivity::class.java).execute()
    }
}
```

`MainActivity` displays a sign-out button, that calls the `signOut()` method of the `IdentityManager`. This will fire the `SignInStateChangeListener.onSignedOut()` event defined in the `AuthenticatorActivity`. In Logcat, you should see the string: `Signing out...`.

`onUserSignedOut()` then calls `showSignIn` which causes the sign-in screen to reappear.

```
package com.YOUR-DOMAIN.android.YOUR-APP-NAME;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
import com.amazonaws.mobile.auth.core.IdentityHandler;
import com.amazonaws.mobile.auth.core.IdentityManager;
import com.amazonaws.mobile.client.AWSMobileClient;

class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        signOutButton.onClick {
            IdentityManager.defaultIdentityManager.signOut()
        }
    }
}
```

iOS - Swift

In the following example, `AWSMobileClient` is instantiated within the `didfinishlaunching` and `open url` blocks in `AppDelegate`, as described in Add User Sign-In (p. 20). If the client does

not find a cached identity from a previous sign-in, it retrieves an unauthenticated “guest” Amazon Cognito Federated Identity ID that is used to access other AWS services. In debug output, look for the string: `Welcome to AWS!`.

If the user already has a cached authenticated identity ID from a previous sign-in, then `AWSMobileClient` will resume the session without an additional sign-in.

When `AWSMobileClient` is instantiated, the app navigates to a Navigation Control hosted in a ViewController whose UIView contains a sign-out button. If the user is not already signed in, the viewDidLoad of the ViewController calls the built-in sign-in UI of the AWS Mobile SDK. A successful sign-in prints the string: `Sign-in succeeded` to debug output.

In the action of the sign-out button, a successful sign-out calls for the sign-in screen to be displayed again.

```
// ViewController.swift

import UIKit

import AWSMobileClient
import AWSAuthCore
import AWSAuthCore
import AWSAuthUI
class ViewController: UIViewController {

    @IBOutlet weak var textfield: UITextField!

    public var identityId: String = ""

    override func viewDidLoad() {
        super.viewDidLoad()

        showSignIn()

    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }

    @IBAction func signOutButtonPress(_ sender: Any) {
            AWSSignInManager.sharedInstance().logout(completionHandler: {(result: Any?,
 error: Error?) in
                self.showSignIn()
             // print("Sign-out Successful: \(signInProvider.getDisplayName)");

            })

    }


    func showSignIn() {
            AWSAuthUIViewController.presentViewController(with:
 self.navigationController!, configuration: nil, completionHandler: {
                (provider: AWSSignInProvider, error: Error?) in
                if error != nil {
                    print("Error occurred: \(String(describing: error))")
                } else {
                    print("Sign-in successful.")

                 }
            })
    }
```

```
}
```

Set Up Facebook Authentication

To use the following Facebook service configuration steps to federate Facebook as a user sign-in provider for AWS services called in your app, try the AWS Mobile HubUser Sign-in feature (p. 348).

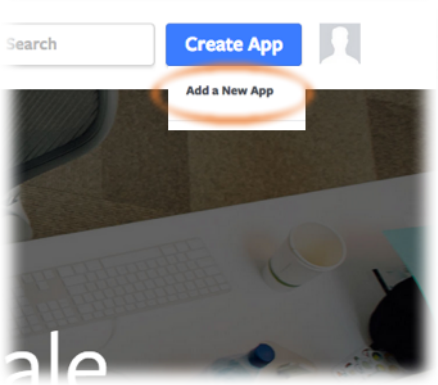
You must first register your application with Facebook by using the Facebook Developers portal.

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.

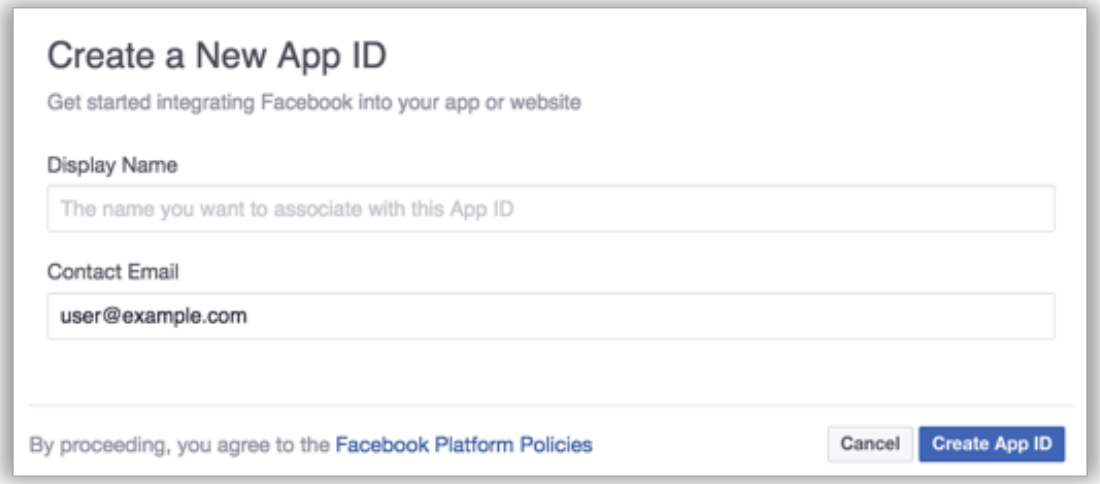
If you already have a Facebook app ID, copy and paste it into the **Facebook App ID** field in the Mobile Hub console, and choose **Save changes**.

To get a Facebook app ID

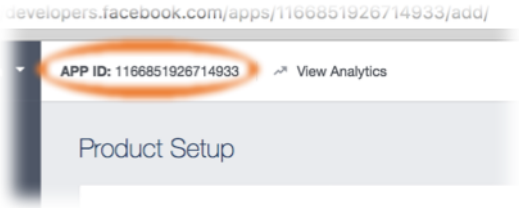
1. In the Facebook Developers portal, sign in with your Facebook credentials.
2. From **Create App**, choose **Add a New App** (note: this menu label will be **My Apps** if you have previously created an app.



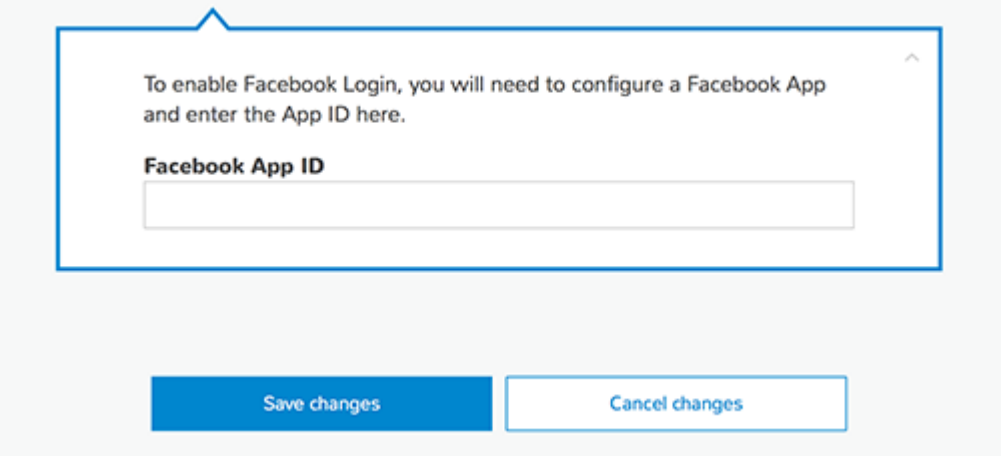
3. If asked, choose the platform of your app that will use Facebook sign-in, and **basic setup**.
4. Type a display name for your app, select a category for your app from the **Category** drop-down list, and then choose **Create App ID**.



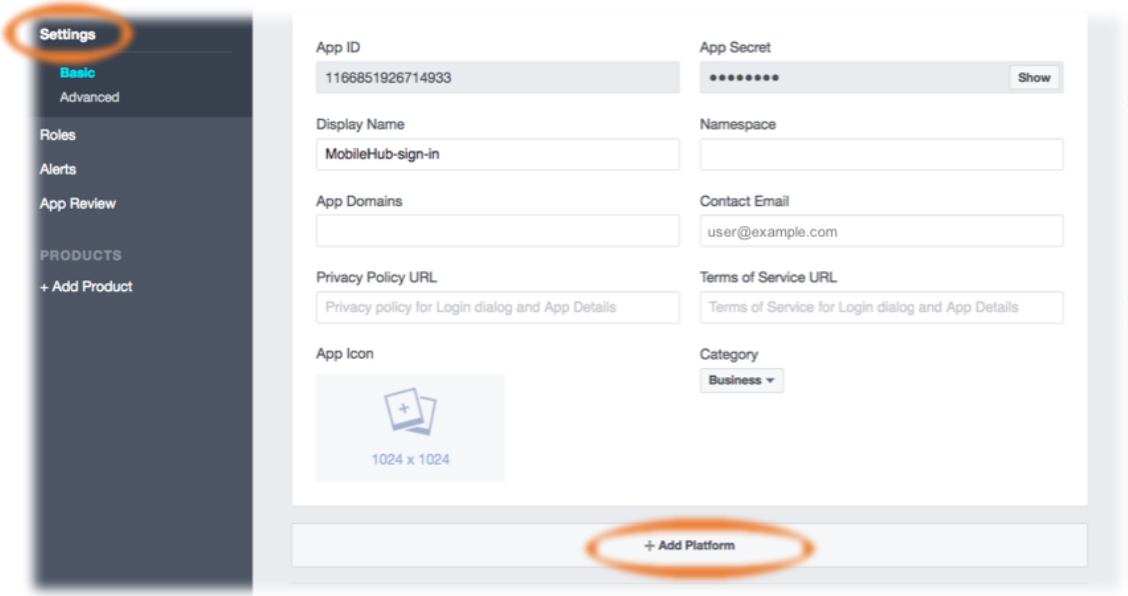
5. Complete the **Security Check** that appears. Your new app then appears in the **Dashboard**.



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



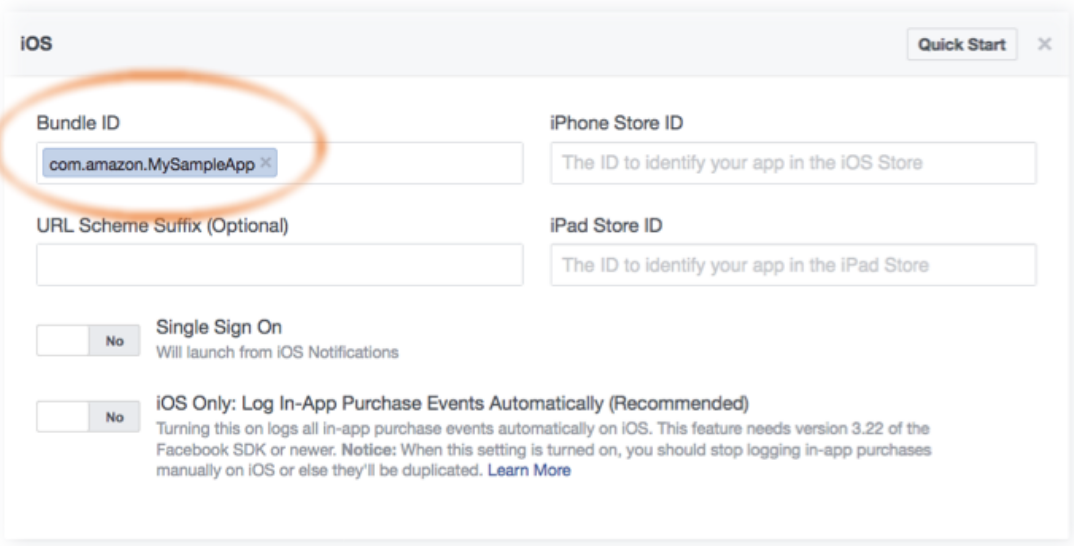
7. In the Facebook Developer portal's left hand navigation list, choose **Settings**, then choose **+ Add Platform**.



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

   For iOS:

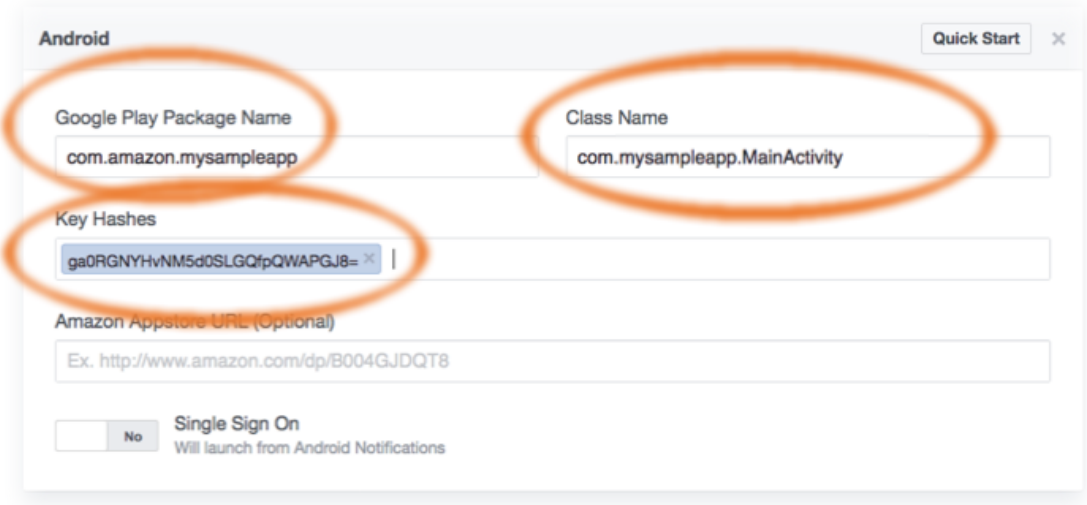
   a. Add your app's **Bundle ID**. (ie. `com.amazon.YourProjectName`). To use the AWS Mobile Hub sample app project, set your this value to `com.amazon.MySampleApp`.



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
```

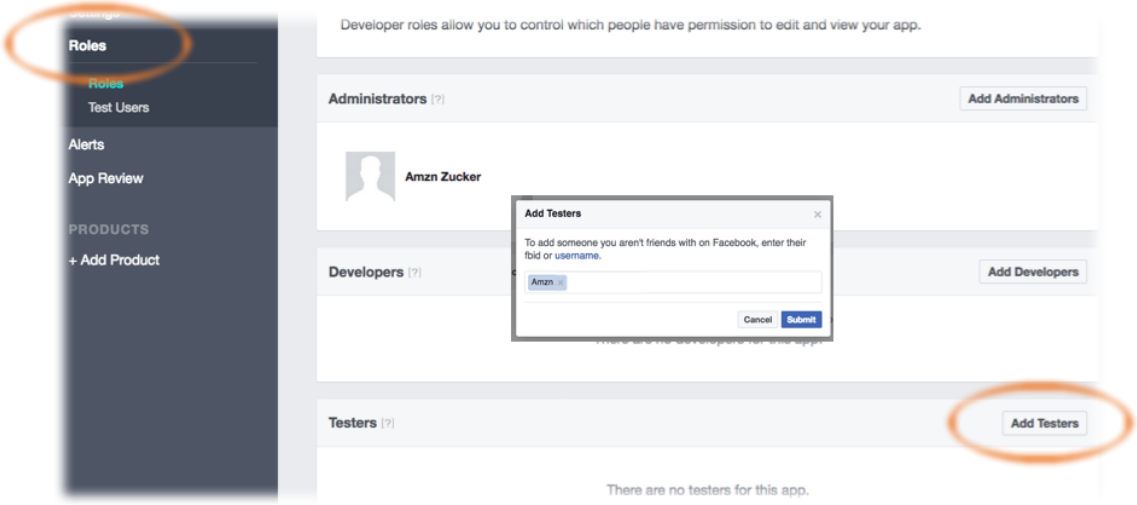
To generate a development key for your Android environment on Windows, run the following command line.

```
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 abel 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.

Set Up Google Authentication

To use the following Google service configuration steps to federate Facebook as a user sign-in provider for AWS services called in your app, try the AWS Mobile HubUser Sign-in feature (p. 348).

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 app, Mobile Hub needs information that comes from Google's setup process.

The following pages detail the Google Sign-In requirements ans steps to integrate Google Sign-In for both iOS and Android apps.

- Create a Google Developers Project and OAuth Web Client ID (p. 162) (required for *all apps* regardless of platform)
- Create an OAuth Android Client ID (p. 168) (required for all Android apps)
- Create an OAuth iOS Client ID (p. 173) (required for all iOS apps)

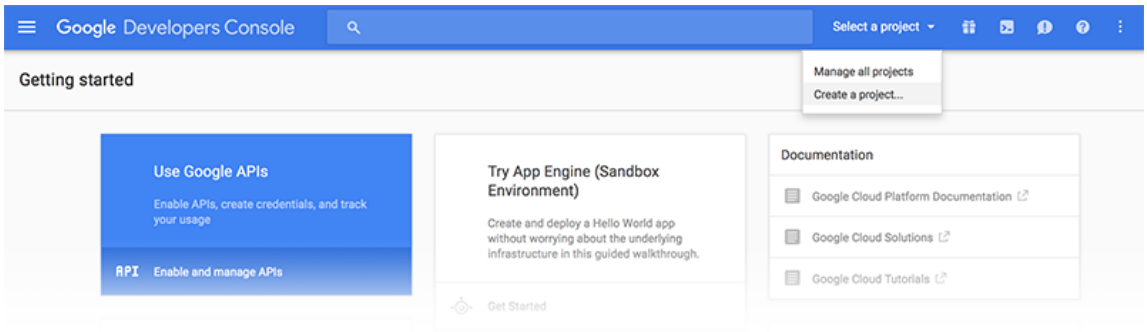
Create a Google Developers Project and OAuth Web Client ID

To enable Google Sign-In in your mobile or web app, create a project in the Google Developers Console. If you are making versions of your mobile app for more than one platform (iOS, Android, or web), create a single Google project to manage Google authentication for all of the platform instances.

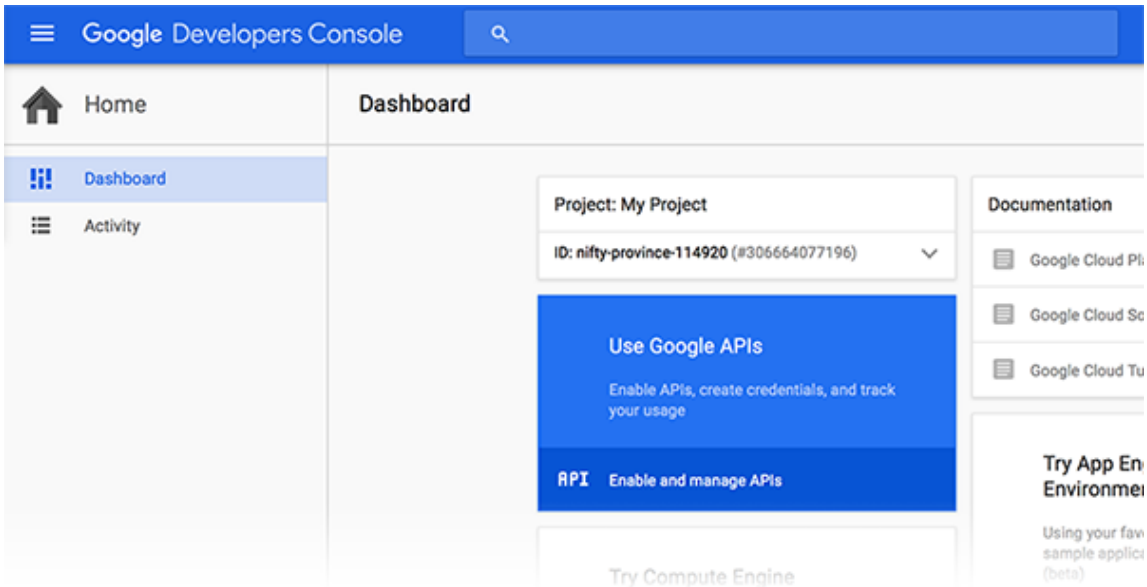
For all platforms, enable the Google+ API for and an OAuth web client ID for your Google project. Amazon Cognito federates the web client ID to enable your app(s) to use Google authentication to grant access to your AWS resources.

To create a Google Developers project and OAuth web client ID

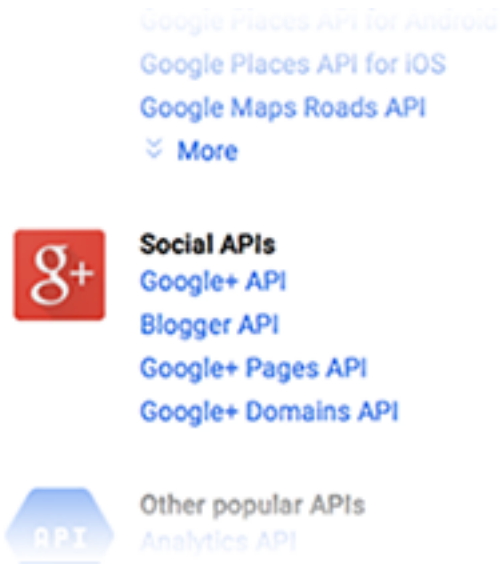
1. Go to the Google Developers Console at https://console.developers.google.com.
2. If you have not created a project yet, choose **Select a project** from the menu bar, and then choose **Create a project...**.



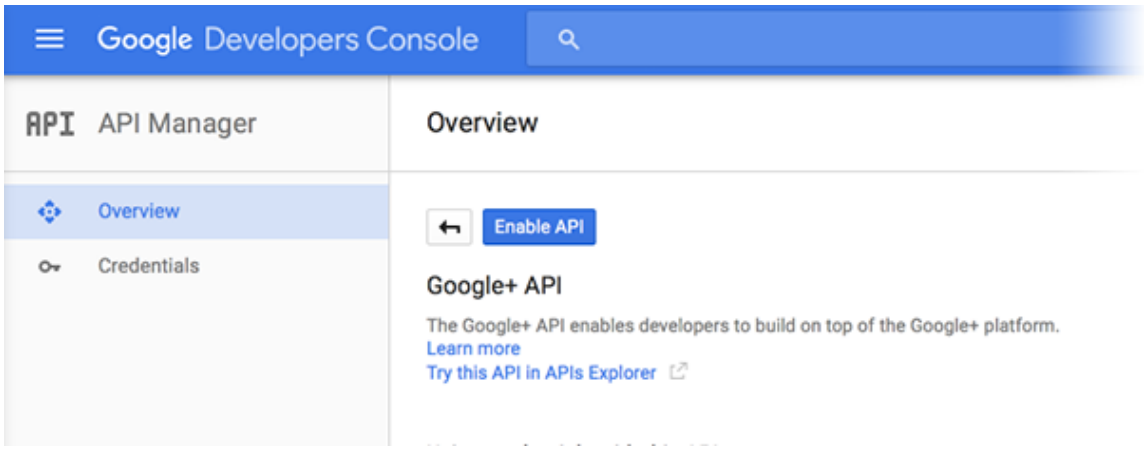
3. Complete the form that is displayed to create your new project.
4. In the **Dashboard** for your project, go to the **Use Google APIs** section and then choose **Enable and manage APIs**.



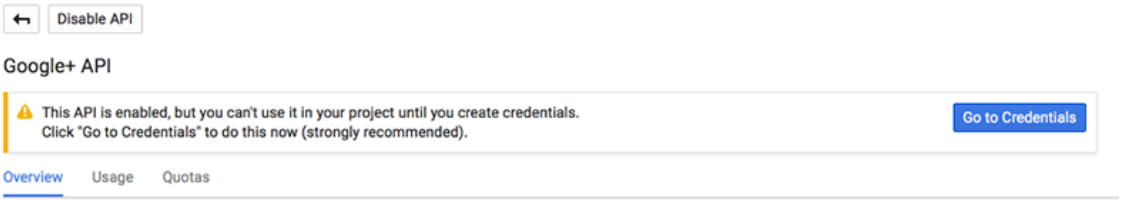
5. In the API Manager, in the **Social APIs** section, choose **Google+ API**.



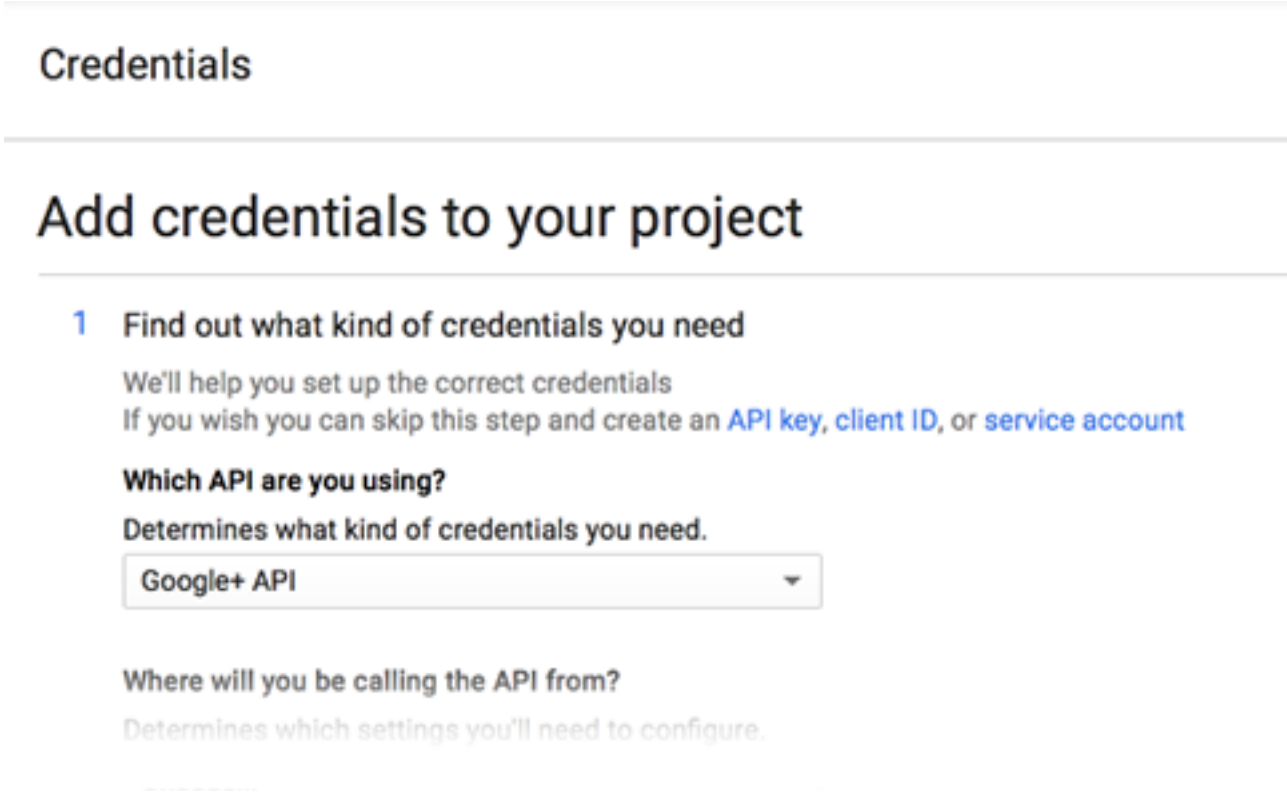
6. In the **Overview** for Google+ API, choose **Enable API**.



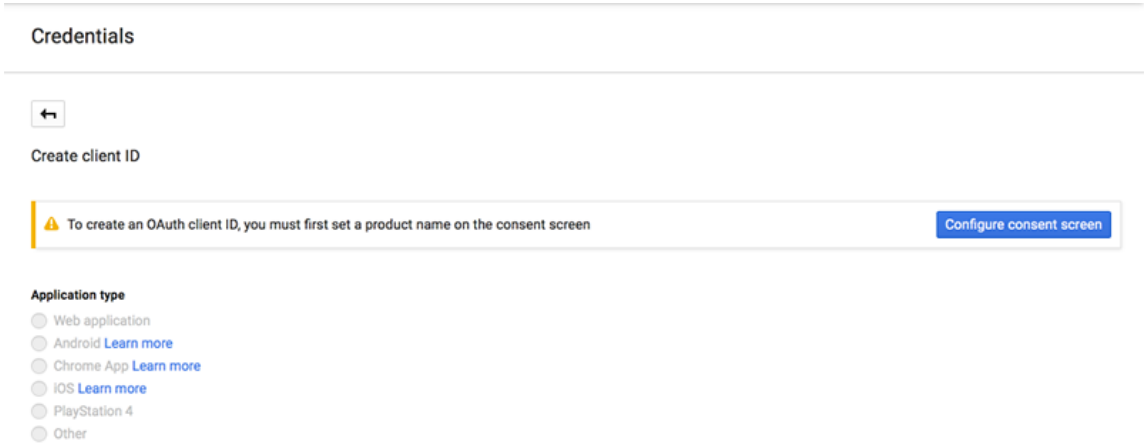
7. A message appears to inform you that the API is enabled but that it requires credentials before you can use it. Choose **Go to Credentials**.



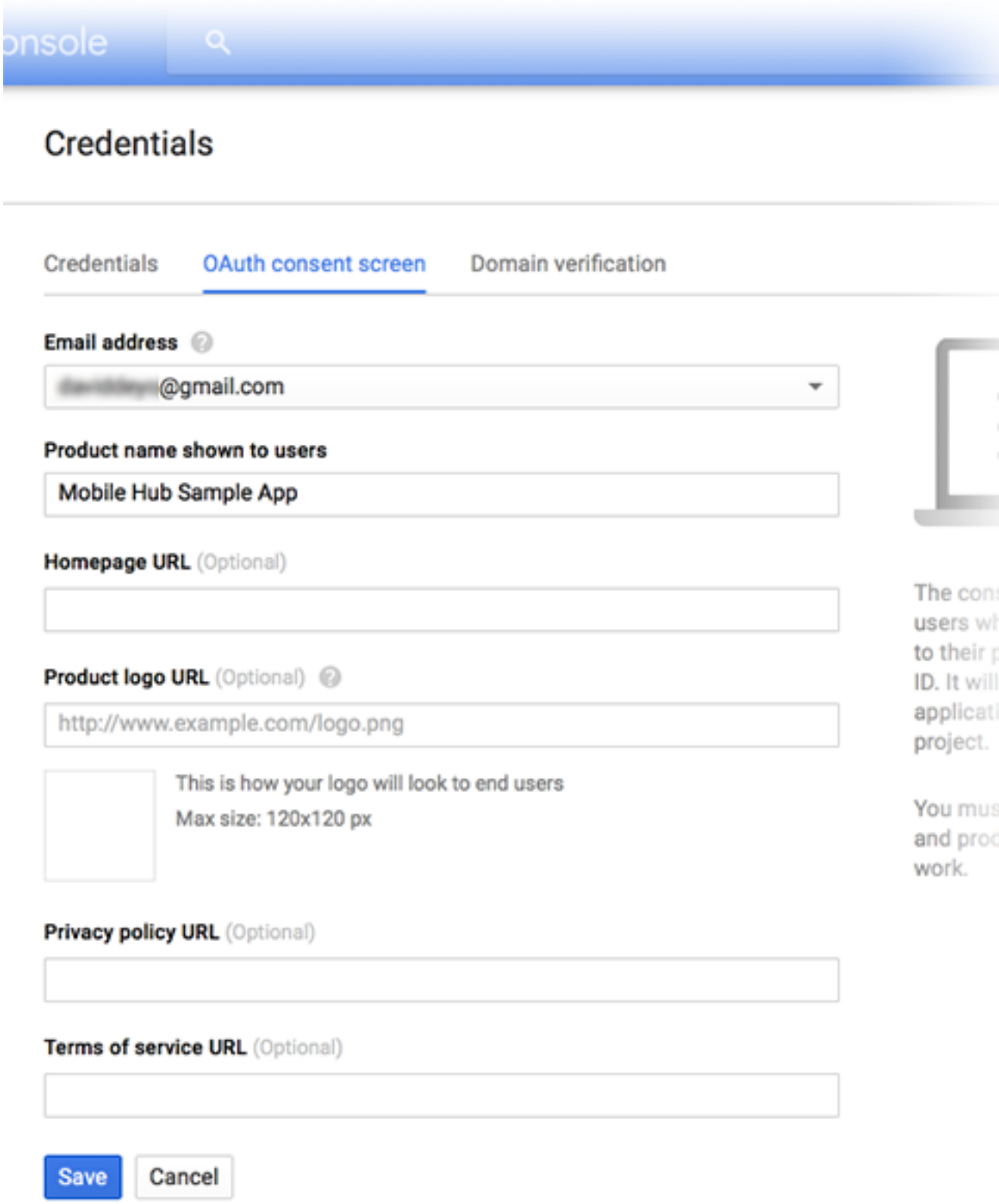
8. Your Mobile Hub sample app authenticates users through Amazon Cognito Identity, so you need an OAuth web application client ID for Amazon Cognito. In **Credentials**, choose **client ID** from the links in the first step.



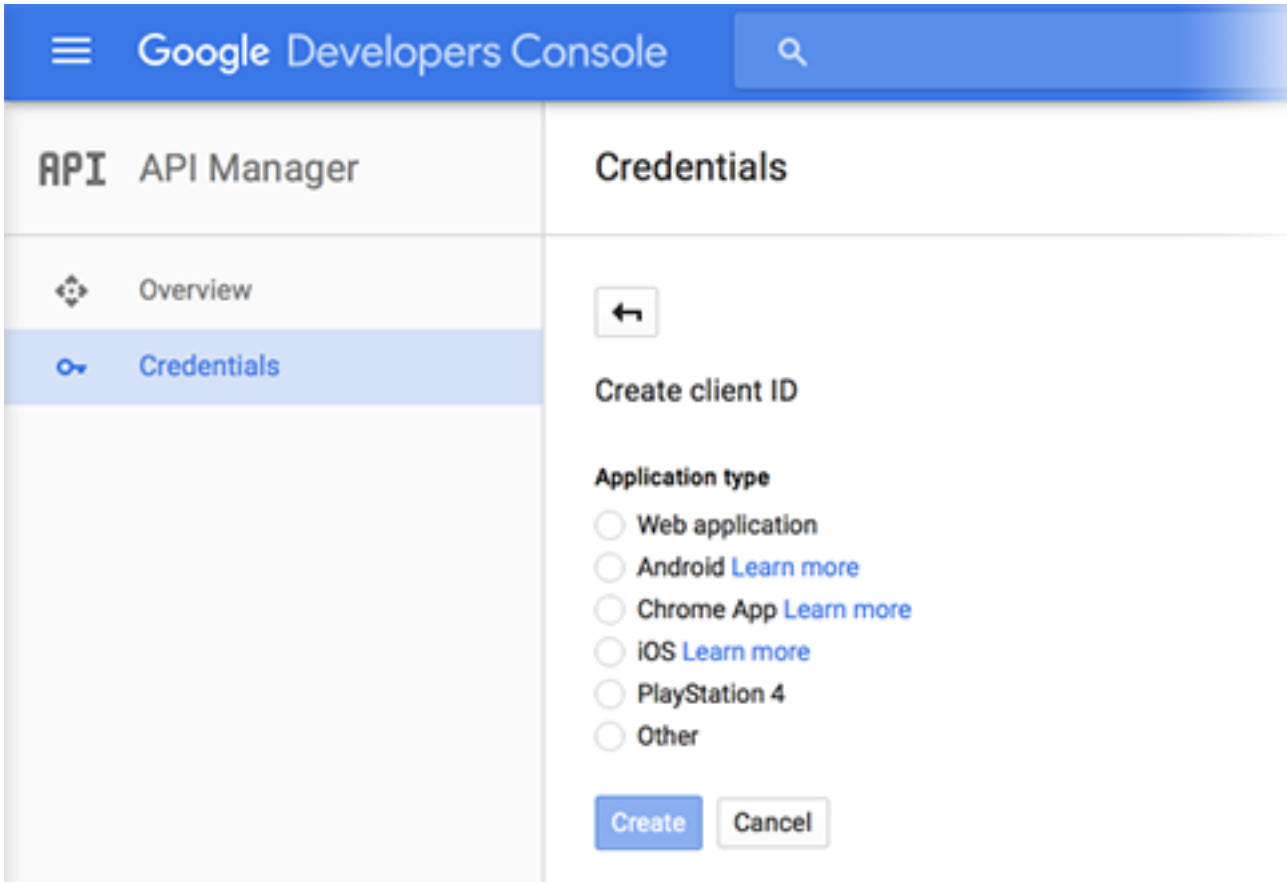
9. A message appears to inform you that you must set a product name. Choose **Configure consent screen**.



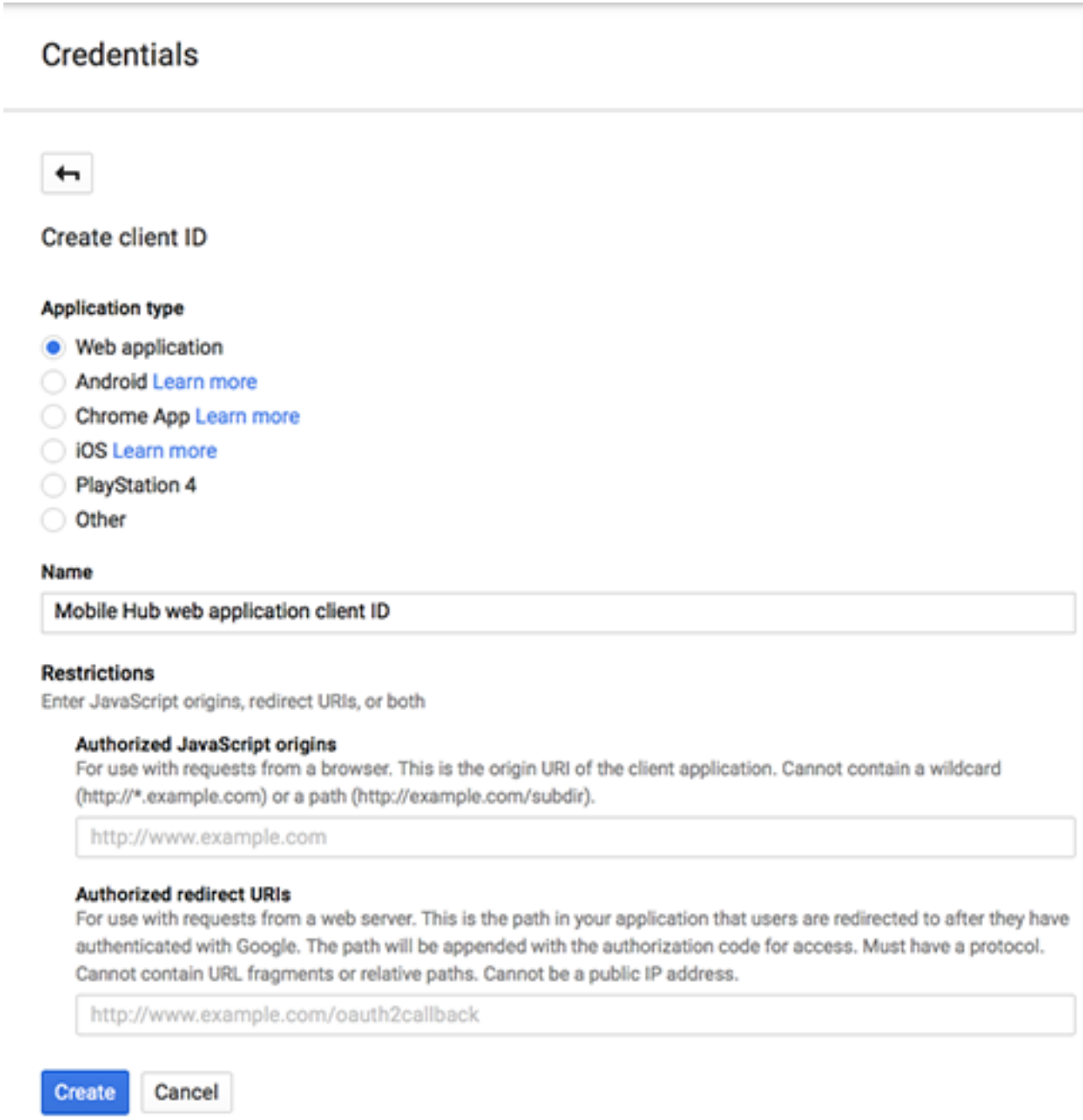
10. In **OAuth consent screen**, enter the name of your app in **Product name shown to users**. Leave the remaining fields blank. Then choose **Save**.



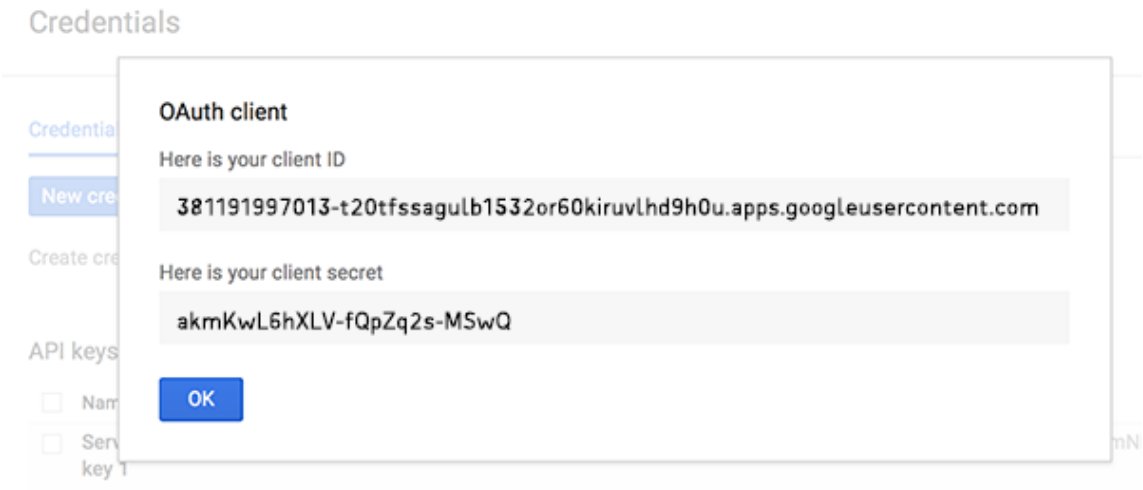
11In **Create client ID**, choose **Web application**.



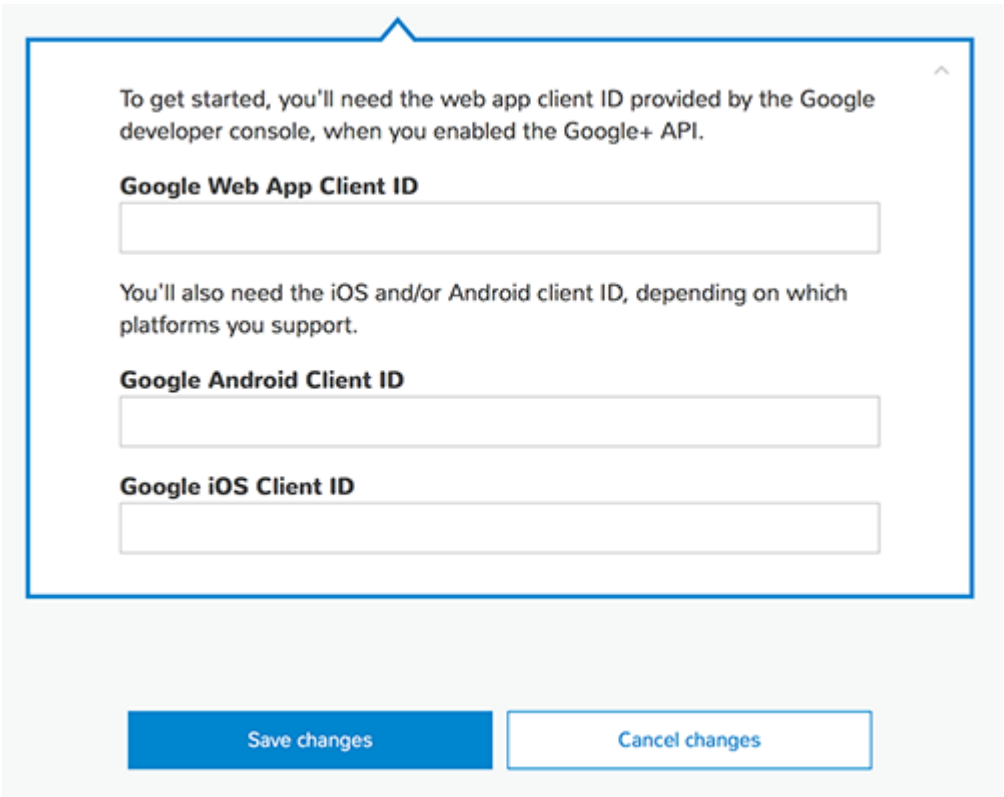
12 In **Name**, enter a name for the web client credentials for your app. Leave the **Authorized JavaScript origins** and **Authorized Redirect URIs** fields blank. Mobile Hub configures this information indirectly through Amazon Cognito Identity integration. Choose **Create**.



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**.



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

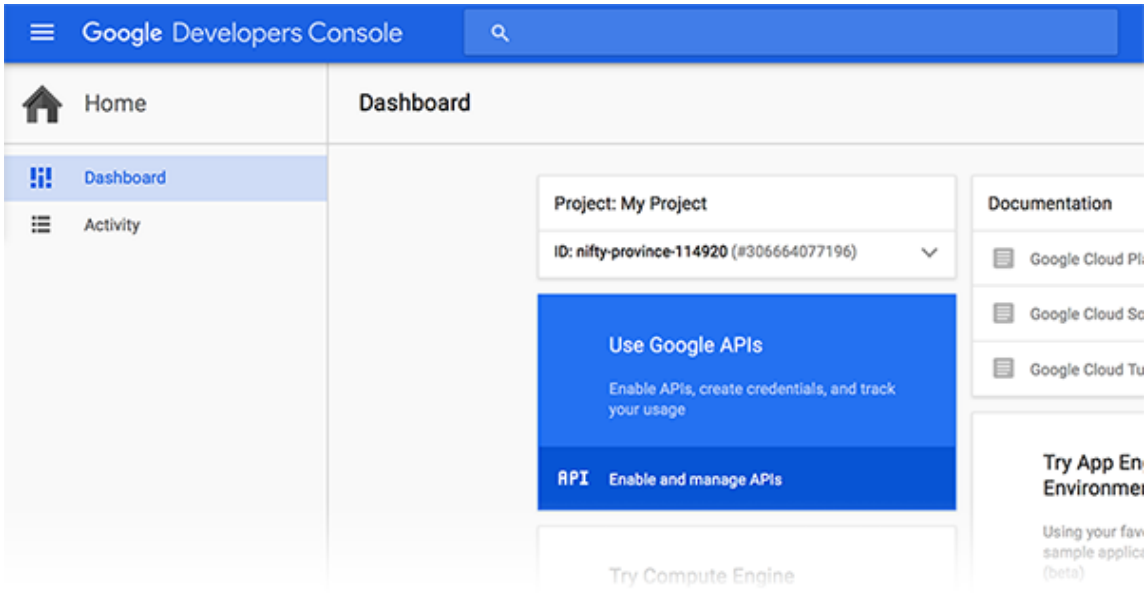


Create an OAuth Android Client ID

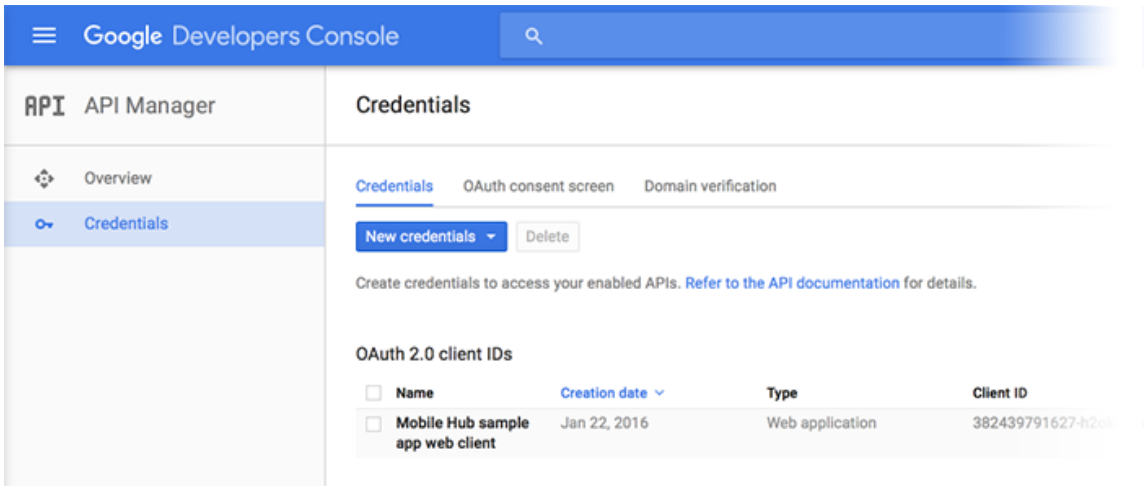
To enable Google Sign-In for your Android app, create an Android OAuth client ID in the Google Developers Console. This enables your app that can access Google APIs directly and manage token lifecycle through Amazon Cognito Identity. This Android OAuth client ID is in addition to the Web application OAuth client ID you created when following steps to Create a Google Developers Project and OAuth Web Client ID (p. 162). You will provide this client ID to Mobile Hub during the Google Sign-In configuration.

To create an OAuth Android client ID

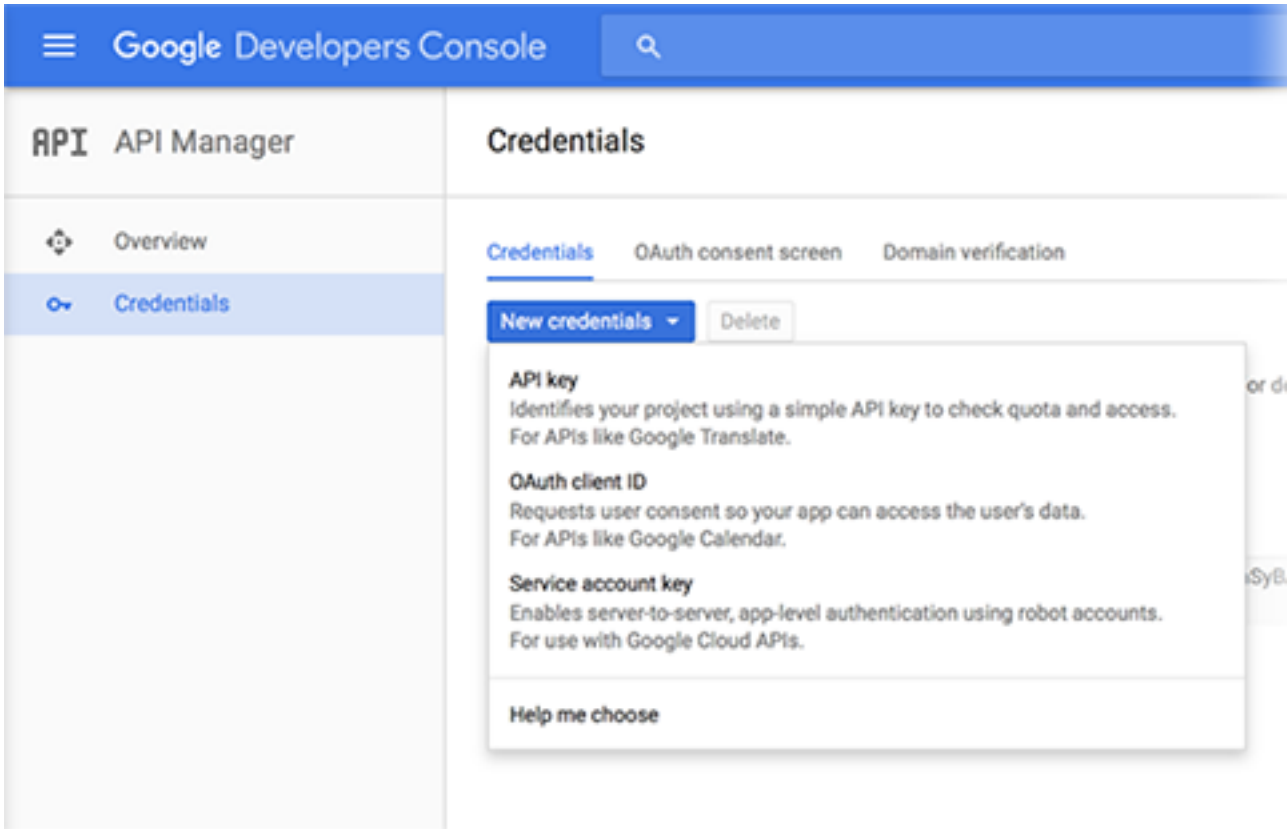
1. Go to the Google Developers Console at https://console.developers.google.com.
2. In the **Dashboard** for your project, go to the **Use Google APIs** section and then choose **Enable and manage APIs**.



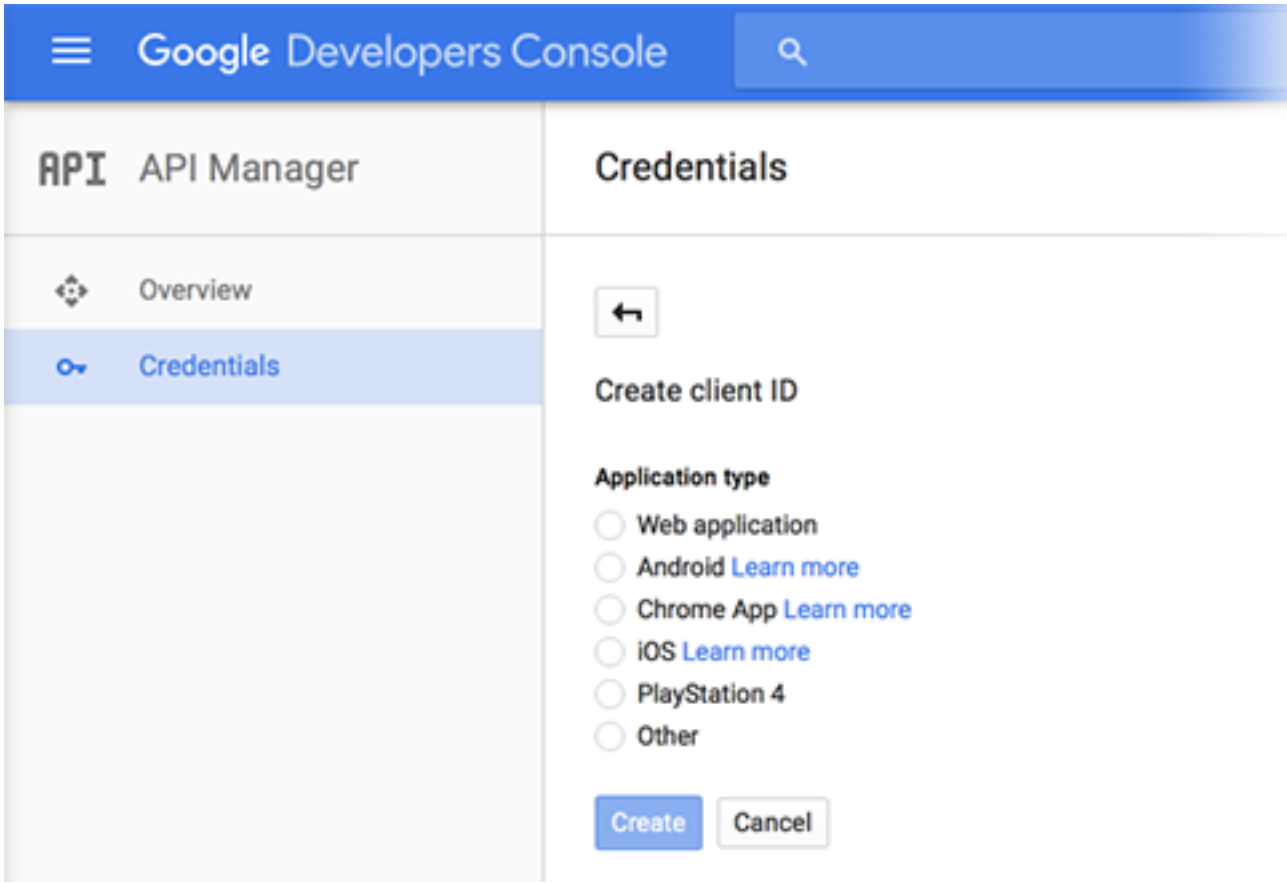
3. In the API Manager, choose **Credentials** in the left side menu.



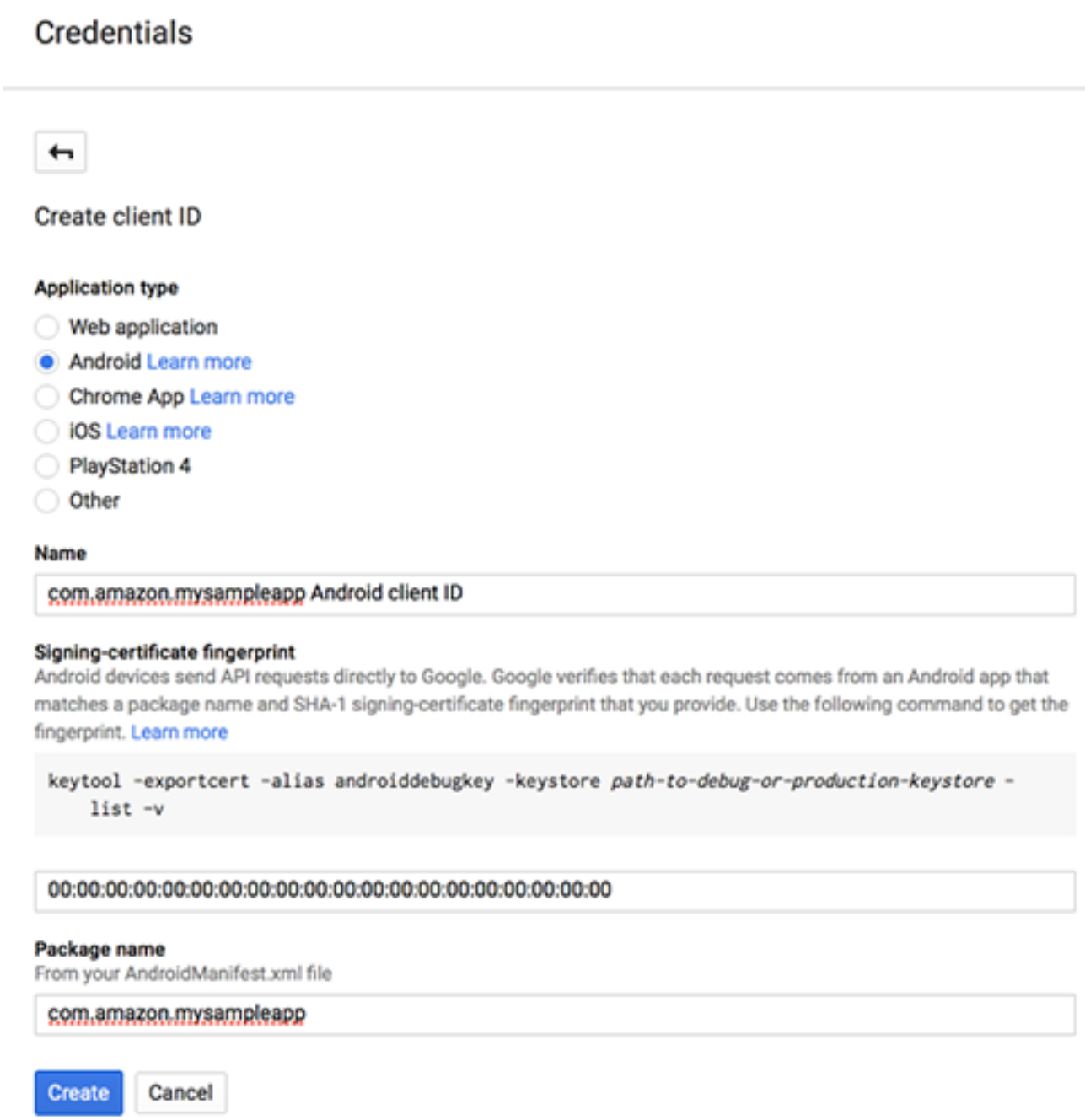
4. Choose **New credentials** and then choose **OAuth client ID**.



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

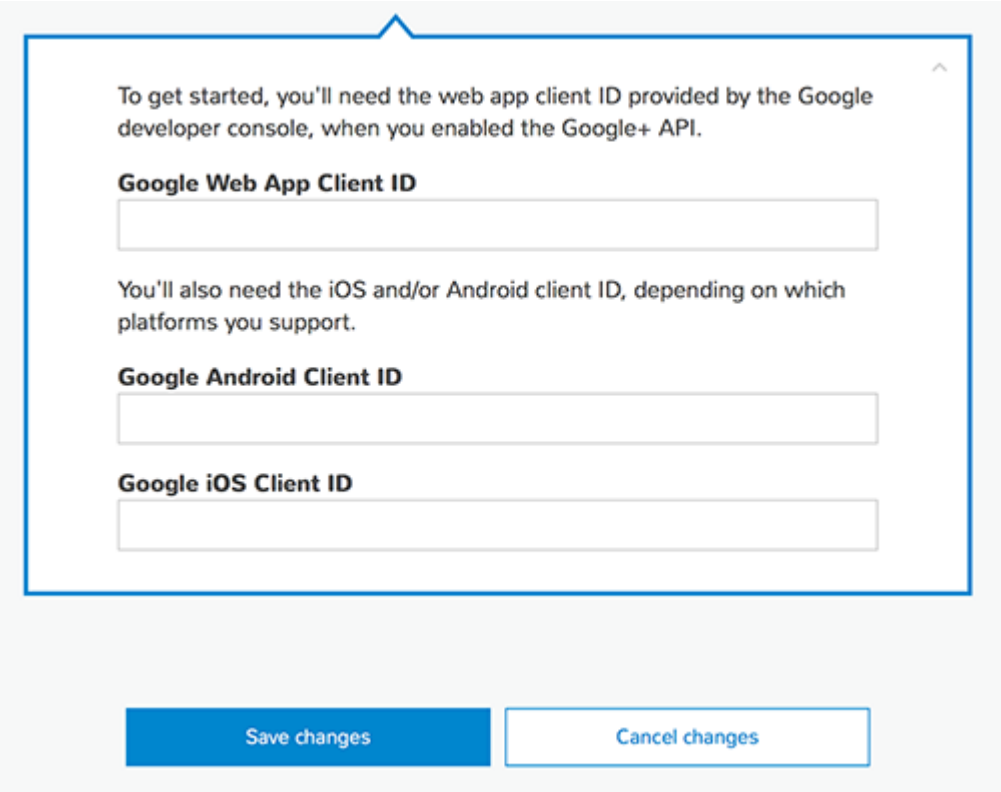


6. In **Name**, enter a name in the format `com.amazon.mysampleapp Android client ID`.
7. In **Signing-certificate fingerprint**, enter the SHA-1 fingerprint. For more information about Google's process for obtaining your SHA-1 fingerprint, see this Google support article.



8. Use your your SHA-1 fingerprint to ensure that your apps APK are associated with your Google app. See instructions at Generate a key and keystore.
9. In **Package name**, enter the package name in the format `com.amazon.YOUR-PACKAGE-NAME`.
10. Choose **Create**.
11. 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**.

12Paste the Android client ID value into the Mobile Hub**Google Android Client ID** field for your project.

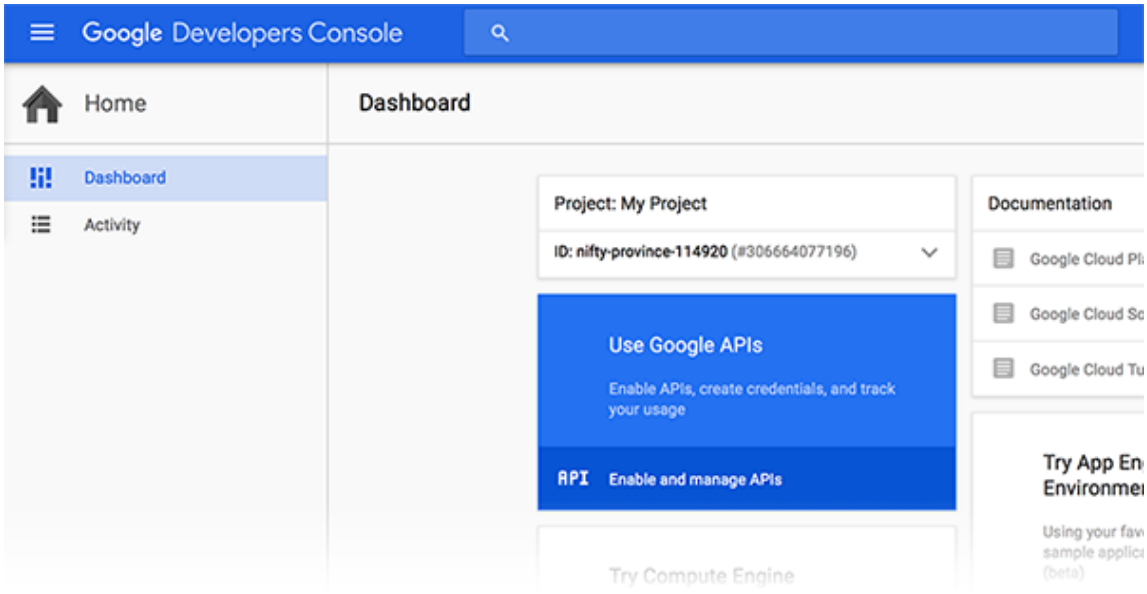


Create an OAuth iOS Client ID

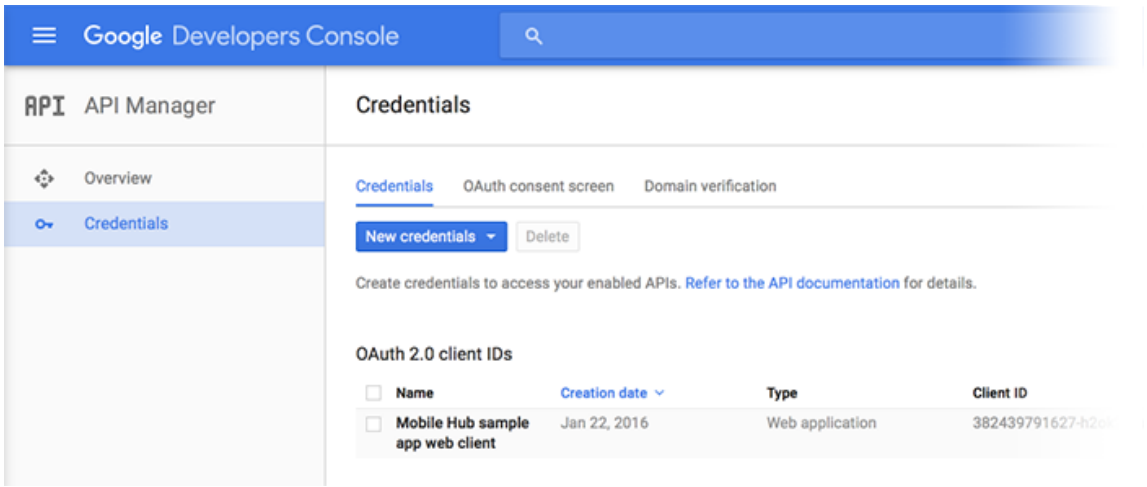
To enable Google Sign-In for your iOS app, create an iOS OAuth client ID in the Google Developers Console. This enables your app to access Google APIs directly and to manage token lifecycle through Amazon Cognito Identity. This iOS OAuth client ID is in addition to the web application OAuth client ID that you created when following steps to Create a Google Developers Project and OAuth Web Client ID (p. 162). You will provide this client ID to Mobile Hub during the Google Sign-In configuration.

To create an OAuth iOS client ID

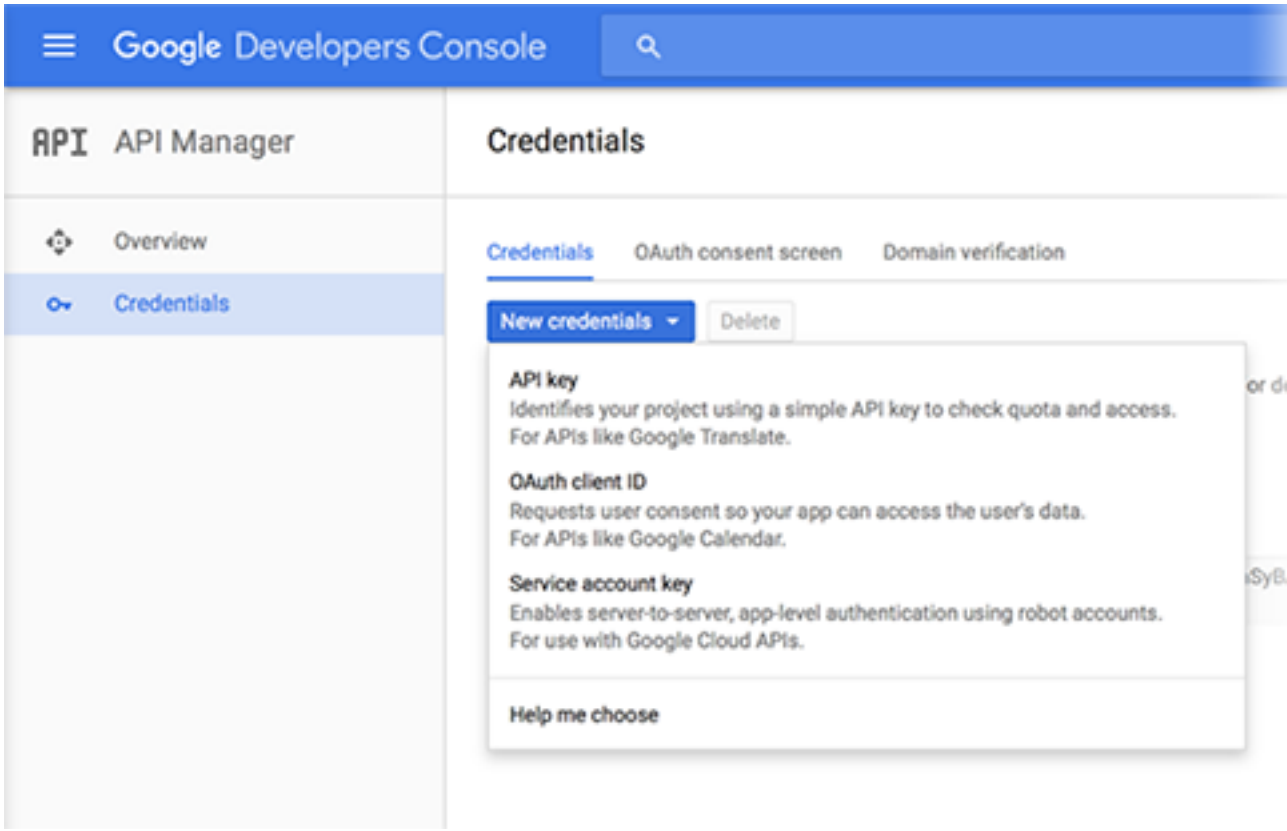
1. Go to the Google Developers Console at https://console.developers.google.com.
2. In the **Dashboard** for your project, go to the **Use Google APIs** section and then choose **Enable and manage APIs**.



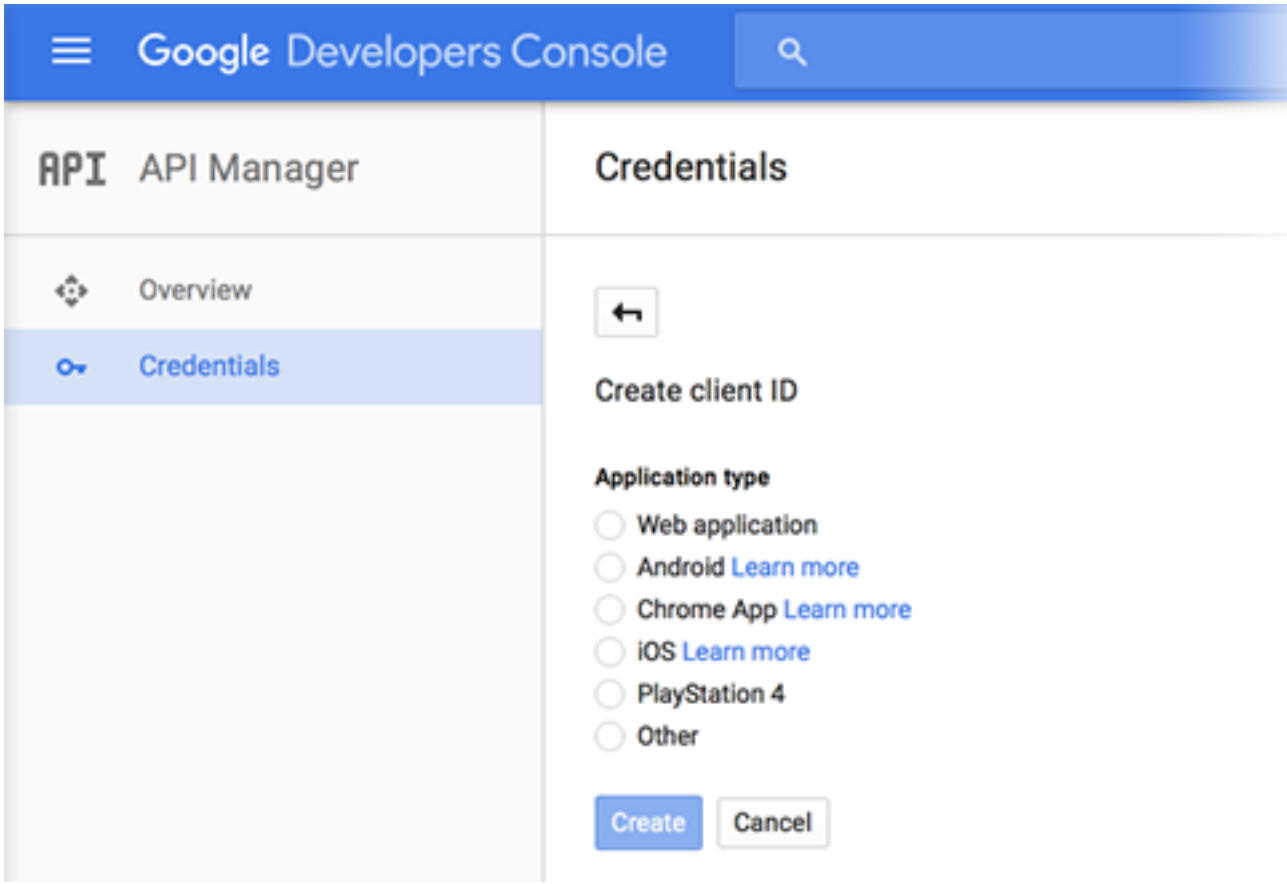
3. In the API Manager, choose **Credentials** in the left side menu.



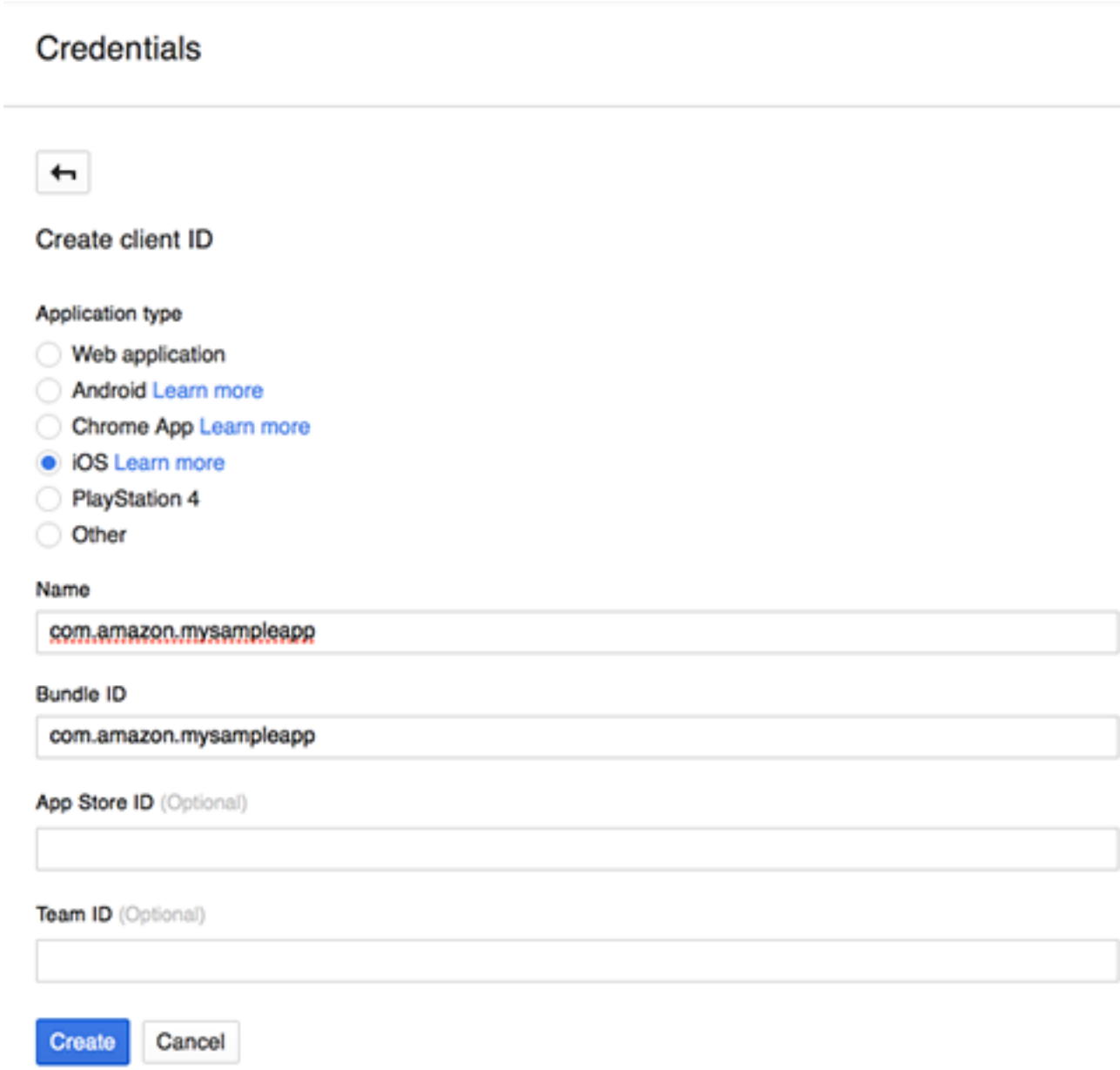
4. Choose **New Credentials** and then choose **OAuth client ID**.



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

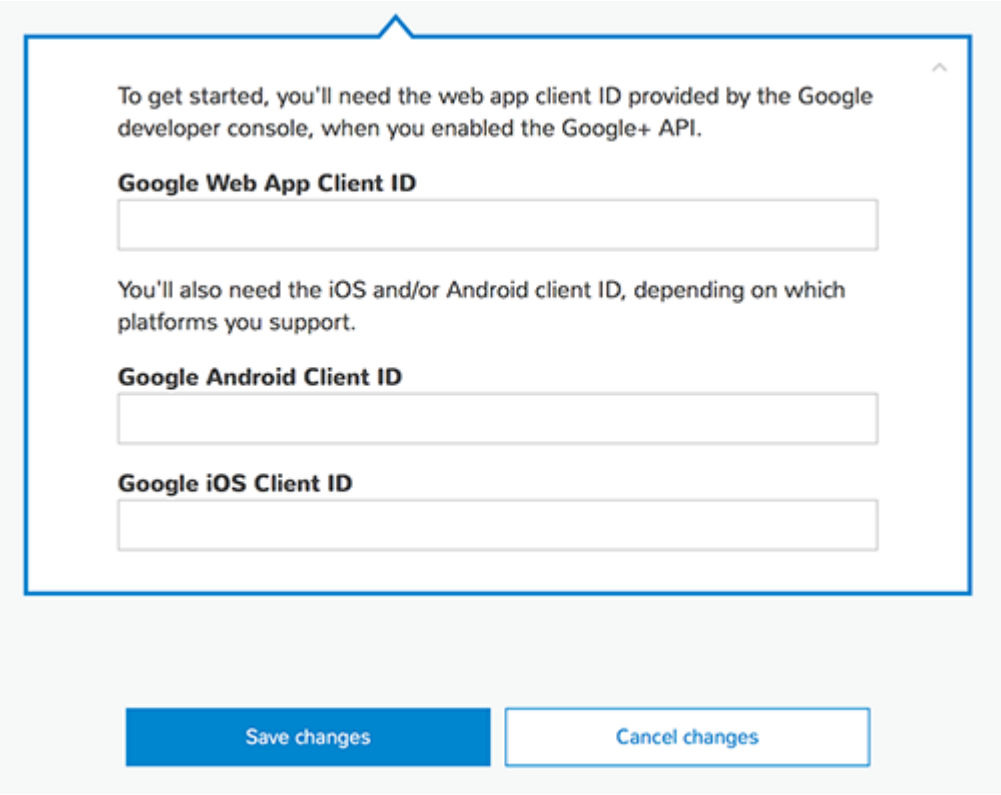


6. In **Name**, enter a name in the format `com.amazon.YOUR-APP-NAME YOUR-iOS-CLIENT-ID`.
7. In **Bundle ID**, enter the bundle name in the format `com.amazon.YOUR-APP-NAME`.



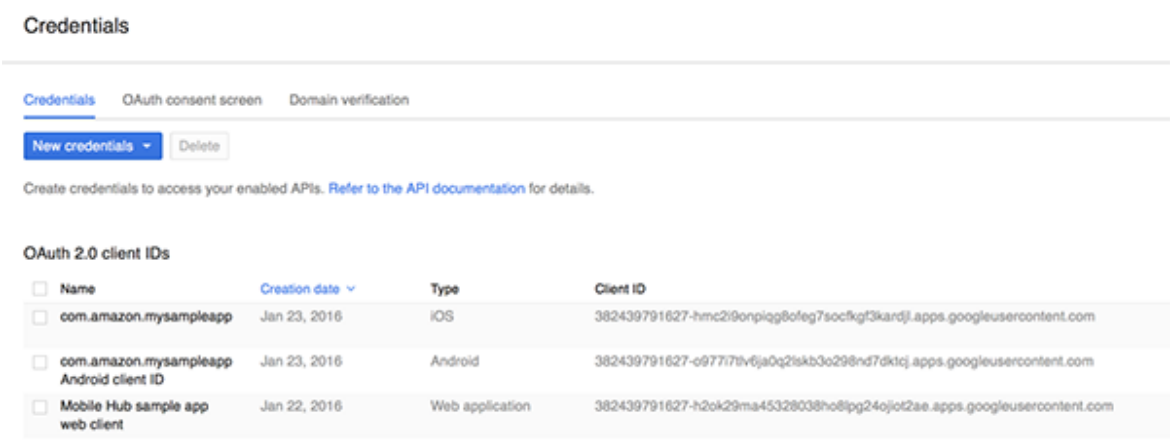
8. Choose **Create**.
9. In the **OAuth client** pop-up, copy and save the value that was generated for your iOS client ID. You will need these values to implement Google Sign-In in your Mobile Hub app. After you copy the client ID, choose **OK**.

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



Verify 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.

Customize the SDK Sign-In UI

By default, the SDK presents sign-in UI for each sign in provider you enable in your Mobile Hub project (Email and Password, Facebook, Google) with a default look and feel. It knows which provider(s) you chose by reading the `awsconfiguration.json` file you downloaded.

To override the defaults, and modify the behavior, look, and feel of the sign-in UI, create an `AuthUIConfiguration` object and set the appropriate properties.

Android - Java

Create and configure an `AuthUIConfiguration` object and set its properties.

- To present the Email and Password user `SignInUI`, set `userPools` to `true`.
- To present Facebook or Google user `SignInUI`, add `signInButton(FacebookButton.class)` or `signInButton(GoogleButton.class)`.
- To change the logo, use the `logoResId`.
- To change the background color, use `backgroundColor`.
- To cancel the sign-in flow, set `.canCancel(true)`.
- To change the font in the sign-in views, use the `fontFamily` method and pass in the string that represents a font family.
- To draw the `backgroundColor` full screen, use `fullScreenBackgroundColor`.

```
import android.app.Activity;
import android.graphics.Color;
import android.os.Bundle;

import com.amazonaws.mobile.auth.facebook.FacebookButton;
import com.amazonaws.mobile.auth.google.GoogleButton;
import com.amazonaws.mobile.auth.ui.AuthUIConfiguration;
import com.amazonaws.mobile.auth.ui.SignInUI;

import com.amazonaws.mobile.client.AWSMobileClient;
import com.amazonaws.mobile.client.AWSStartupHandler;
import com.amazonaws.mobile.client.AWSStartupResult;

public class YourMainActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        AWSMobileClient.getInstance().initialize(this, new AWSStartupHandler() {
            @Override
            public void onComplete(final AWSStartupResult awsStartupResult) {
                AuthUIConfiguration config =
                    new AuthUIConfiguration.Builder()
                        .userPools(true)  // true? show the Email and Password UI
                        .signInButton(FacebookButton.class) // Show Facebook button
                        .signInButton(GoogleButton.class) // Show Google button
                        .logoResId(R.drawable.mylogo) // Change the logo
                        .backgroundColor(Color.BLUE) // Change the backgroundColor
                        .isBackgroundColorFullScreen(true) // Full screen
 backgroundColor the backgroundColor full screenff
                        .fontFamily("sans-serif-light") // Apply sans-serif-light as
 the global font
                        .canCancel(true)
                        .build();
                SignInUI signinUI = (SignInUI)
 AWSMobileClient.getInstance().getClient(YourMainActivity.this, SignInUI.class);
                signinUI.login(YourMainActivity.this,
 YourNextActivity.class).authUIConfiguration(config).execute();
            }
        }).execute();
    }
}
```

Android - Kotlin

Create and configure an `AuthUIConfiguration` object and set its properties.

- To present the Email and Password user `SignInUI`, set `userPools` to `true`.
- To present Facebook or Google user `SignInUI`, add `signInButton(FacebookButton.class)` or `signInButton(GoogleButton.class)`.
- To change the logo, use the `logoResId`.
- To change the background color, use `backgroundColor`.
- To cancel the sign-in flow, set `.canCancel(true)`.
- To change the font in the sign-in views, use the `fontFamily` method and pass in the string that represents a font family.
- To draw the `backgroundColor` full screen, use `fullScreenBackgroundColor`.

```
import android.app.Activity;
import android.graphics.Color;
```

```
import android.os.Bundle;

import com.amazonaws.mobile.auth.facebook.FacebookButton;
import com.amazonaws.mobile.auth.google.GoogleButton;
import com.amazonaws.mobile.auth.ui.AuthUIConfiguration;
import com.amazonaws.mobile.auth.ui.SignInUI;

import com.amazonaws.mobile.client.AWSMobileClient;
import com.amazonaws.mobile.client.AWSStartupHandler;
import com.amazonaws.mobile.client.AWSStartupResult;

class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState : Bundle?) {
        super.onCreate()
        AWSMobileClient.getInstance().initialize(this) {
            val config = AuthUIConfiguration.Builder()
                    .userPools(true) // show the Email and Password UI
                    .signInButton(FacebookButton.class) // Show Facebook
                    .signInButton(GoogleButton.class) // Show Google
                    .logoResId(R.drawable.mylogo) // Change the logo
                    .backgroundColor(Color.BLUE) // Change the background color
                    .isBackgroundColorFullScreen(true) // Full screen background color
                    .fontFamily("sans-serif-light") // font
                    .canCancel(true) // Add a cancel/back button
                    .build()
            val signInUI = AWSMobileClient.getInstance().getClient(this@MainActivity,
 SignInUI::class.java) as SignInUI
            signInUI.login(this@MainActivity,
 NextActivity::class.java).authUIConfiguration(config).execute()
        }.execute()
    }
}
```

iOS - Swift

Create and configure an `AWSAuthUIConfiguration` object and set its properties.

Create and configure an `AuthUIConfiguration` object.

- To present the Email and Password user `SignInUI`, set et `enableUserPoolsUI` to `true`.
- To present Facebook or Google user `SignInUI`, add `.addSignInButtonView(class: AWSFacebookSignInButton.self)` or `.addSignInButtonView(class: AWSFacebookSignInButton.self)`.
- To change the logo, use `logoImage`.
- To change the background color, use `backgroundColor`.
- To cancel the sign-in flow, use `canCancel`.
- To change the font in the sign-in views, use the `font` property and pass in the `UIFont` object that represents a font family.
- To draw the `backgroundColor` full screen, use `fullScreenBackgroundColor`.

```
import UIKit
import AWSAuthUI
import AWSMobileClient
import AWSUserPoolsSignIn
import AWSFacebookSignIn
import AWSGoogleSignIn

class SampleViewController: UIViewController {
    override func viewDidLoad() {
        super.viewDidLoad()
```

```
        if !AWSSignInManager.sharedInstance().isLoggedIn {
            presentAuthUIViewController()
        }
    }

    func presentAuthUIViewController() {
        let config = AWSAuthUIConfiguration()
        config.enableUserPoolsUI = true
        config.addSignInButtonView(class: AWSFacebookSignInButton.self)
        config.addSignInButtonView(class: AWSGoogleSignInButton.self)
        config.backgroundColor = UIColor.blue
        config.font = UIFont (name: "Helvetica Neue", size: 20)
        config.isBackgroundColorFullScreen = true
        config.canCancel = true

        AWSAuthUIViewController.presentViewController(
            with: self.navigationController!,
            configuration: config, completionHandler: { (provider: AWSSignInProvider,
 error: Error?) in
                if error == nil {
                    // SignIn succeeded.
                } else {
                    // end user faced error while loggin in, take any required action
 here.
                }
        })
    }
}
```

How To: File Storage with Amazon S3

Just Getting Started?	Use streamlined steps (p. 66) to install the SDK and integrate Amazon S3.

Or, use the contents of this page if your app will integrate existing AWS services.

This section provides information on the steps for achieving specific tasks for integrating your Amazon S3 into your Android and iOS apps.

Topics

- How to Integrate Your Existing Bucket (p. 182)
- Transfer Files and Data Using TransferUtility and Amazon S3 (p. 194)
- Amazon S3 Pre-Signed URLs: For Background Transfer (p. 219)
- Amazon S3 Server-Side Encryption Support in iOS (p. 222)
- iOS: Amazon S3 TransferManager for iOS (p. 223)

How to Integrate Your Existing Bucket

Just Getting Started?	Use streamlined steps (p. 66) to install the SDK and integrate Amazon S3.

Or, use the contents of this page if your app will integrate existing AWS services.

The following steps include:

- Set up short-lived credentials for accessing your AWS resources using a Cognito Identity Pool.
- Create an AWS Mobile configuration file that ties your app code to your bucket.

Set Up Your Backend

If you already have a Cognito Identity Pool and have its unauthenticated IAM role set up with read/write permissions on the S3 bucket, you can skip to Get Your Bucket Name and ID (p. 183).

Create or Import the Amazon Cognito Identity Pool

1. Go to Amazon Cognito Console and choose **Manage Federated Identities**.
2. Choose **Create new Identity pool** on the top left of the console.
3. Type a name for the Identity pool, select **Enable access to unauthenticated identities** under the **Unauthenticated Identities** section, and then choose **Create pool** on the bottom right.
4. Expand the **View Details** section to see the two roles that are to be created to enable access to your bucket. Copy and keep the Unauthenticated role name, in the form of `Cognito_<IdentityPoolName>Unauth_Role`, for use in a following configuration step. Choose **Allow** on the bottom right.
5. In the code snippet labeled **Get AWSCredentials** displayed by the console, copy the Identity Pool ID and the Region for use in a following configuration step.

Set up the required Amazon IAM permissions

1. Go to Amazon IAM Console and choose **Roles**.
2. Choose the unauthenticated role whose name you copied in a previous step.
3. Choose **Attach Policy**, select the `AmazonS3FullAccess` policy, and then choose **Attach Policy** to attach it to the role.

Note	The `AmazonS3FullAccess` policy will grant users in the identity pool full access to all buckets and operations in Amazon S3. In a real app, you should restrict users to only have access to the specific resources they need. For more information, see Amazon S3 Security Considerations.

Get Your Bucket Name and ID

1. Go to Amazon S3 Console and select the bucket you want to integrate.
2. Copy and keep the bucket name value from the breadcrumb at the top of the console, for use in a following step.
3. Copy and keep the bucket's region, for use in a following step.

Connect to Your Backend

Create the awsconfiguration.json file

1. Create a file with name `awsconfiguration.json` with the following contents:

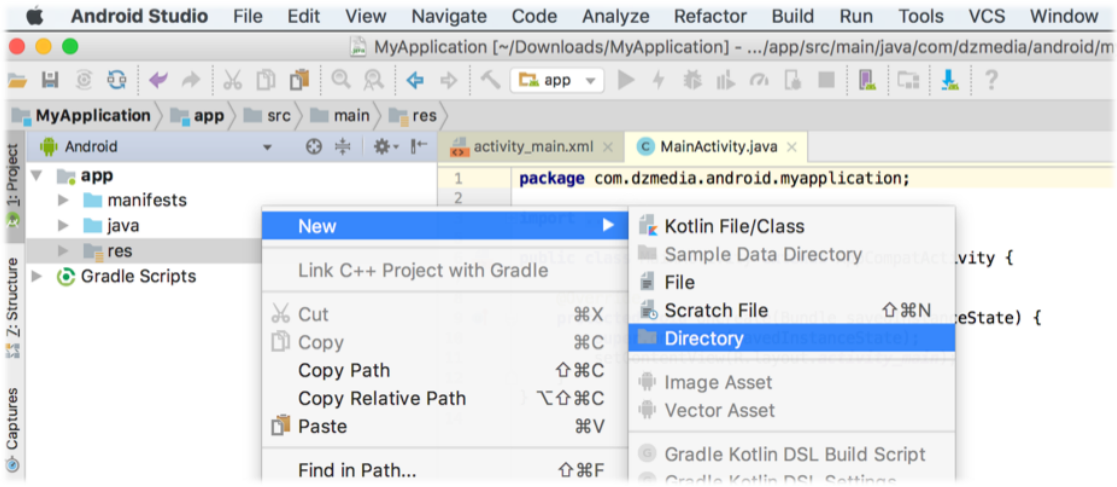
```
{
    "Version": "1.0",
    "CredentialsProvider": {
        "CognitoIdentity": {
            "Default": {
                "PoolId": "COGNITO-IDENTITY-POOL-ID",
                "Region": "COGNITO-IDENTITY-POOL-REGION"
            }
        }
    },
    "IdentityManager" : {
      "Default" : { }
    },
    "S3TransferUtility": {
        "Default": {
            "Bucket": "S3-BUCKET-NAME",
            "Region": "S3-REGION"
        }
    }
}
```

2. Make the following changes to the configuration file.
   - Replace the `COGNITO-IDENTITY-POOL-ID` with the identity pool ID.
   - Replace the `COGNITO-IDENTITY-POOL-REGION` with the region the identity pool was created in.
   - Replace the `S3-BUCKET-NAME` with the name of your bucket.
   - Replace the `S3-REGION` with the region your bucket was created in.

Add the awsconfiguration.json file to your app

Android - Java

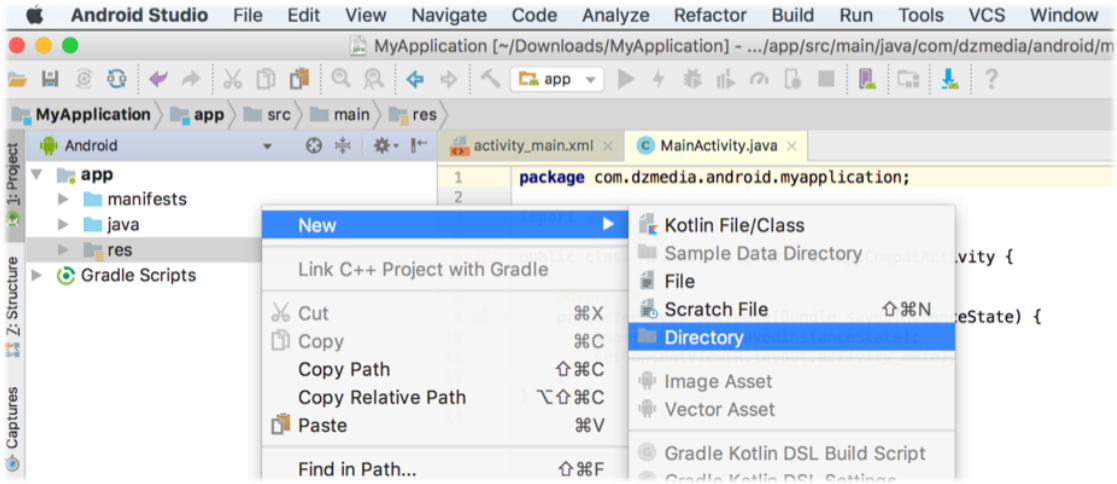
In the Android Studio Project Navigator, right-click your app's `res` folder, and then choose **New > Directory**. Type `raw` as the directory name and then choose **OK**.



Drag the `awsconfiguration.json` you created into the `res/raw` folder. Android gives a resource ID to any arbitrary file placed in this folder, making it easy to reference in the app.

Android - Kotlin

In the Android Studio Project Navigator, right-click your app's `res` folder, and then choose **New > Directory**. Type `raw` as the directory name and then choose **OK**.



Drag the `awsconfiguration.json` you created into the `res/raw` folder. Android gives a resource ID to any arbitrary file placed in this folder, making it easy to reference in the app.

iOS - Swift

Drag the `awsconfiguration.json` into the Xcode Project Navigator folder containing `Info.plist`. Choose **Copy items** and **Create groups** in the options dialog.

Add the SDK to your App

Android - Java

Set up AWS Mobile SDK components as follows:

1. Add the following to `app/build.gradle`:

```
dependencies {
   implementation ('com.amazonaws:aws-android-sdk-mobile-client:2.6.+@aar')
 { transitive = true }
   implementation 'com.amazonaws:aws-android-sdk-s3:2.6.+'
   implementation 'com.amazonaws:aws-android-sdk-cognito:2.6.+'
}
```

   Perform a *Gradle Sync* to download the AWS Mobile SDK components into your app

2. Add the following to `AndroidManifest.xml`:

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />

<application ... >

   <!- . . . ->

   <service
 android:name="com.amazonaws.mobileconnectors.s3.transferutility.TransferService"
 android:enabled="true" />

   <!- . . . ->

</application>
```

3. For each Activity where you make calls to perform user file storage operations, import the following packages.

```
import com.amazonaws.mobile.config.AWSConfiguration;
import com.amazonaws.mobileconnectors.s3.transferutility.*;
```

4. Add the following code to the `onCreate` method of your main or startup activity. This will establish a connection with AWS Mobile. `AWSMobileClient` is a singleton that will be an interface for your AWS services.

```
import com.amazonaws.mobile.client.AWSMobileClient;

public class YourMainActivity extends Activity {
 @Override
 protected void onCreate(Bundle savedInstanceState) {
     super.onCreate(savedInstanceState);

     AWSMobileClient.getInstance().initialize(this).execute();
  }
}
```

Android - Kotlin

Set up AWS Mobile SDK components as follows:

1. Add the following to `app/build.gradle`:

```
dependencies {
   implementation ('com.amazonaws:aws-android-sdk-mobile-client:2.6.+@aar')
 { transitive = true }
   implementation 'com.amazonaws:aws-android-sdk-s3:2.6.+'
   implementation 'com.amazonaws:aws-android-sdk-cognito:2.6.+'
}
```

   Perform a *Gradle Sync* to download the AWS Mobile SDK components into your app

2. Add the following to `AndroidManifest.xml`:

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />

<application ... >

   <!- . . . ->

   <service
 android:name="com.amazonaws.mobileconnectors.s3.transferutility.TransferService"
 android:enabled="true" />

   <!- . . . ->

</application>
```

3. For each Activity where you make calls to perform user file storage operations, import the following packages.

```
import com.amazonaws.mobile.config.AWSConfiguration;
import com.amazonaws.mobileconnectors.s3.transferutility.*;
```

4. Add the following code to the `onCreate` method of your main or startup activity. This will establish a connection with AWS Mobile. `AWSMobileClient` is a singleton that will be an interface for your AWS services.

```
import com.amazonaws.mobile.client.AWSMobileClient;

class MainActivity : Activity() {
  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    AWSMobileClient.getInstance().initialize(this).execute()
  }
}
```

iOS - Swift

Set up AWS Mobile SDK components as follows:

1. Add the following to `Podfile` that you configure to install the AWS Mobile SDK:

```
platform :ios, '9.0'

   target :'YOUR-APP-NAME' do
      use_frameworks!

      pod 'AWSMobileClient', '~> 2.6.13'  # For AWSMobileClient
      pod 'AWSS3', '~> 2.6.13'            # For file transfers
      pod 'AWSCognito', '~> 2.6.13'       # For data sync

      # other pods

   end
```

   Run `pod install --repo-update` before you continue.

   If you encounter an error message that begins "`[!] Failed to connect to GitHub to update the CocoaPods/Specs . . .`", and your internet connectivity is working, you may need to update openssl and Ruby.
2. Add the following imports to the classes that perform user file storage operations:

```
import AWSCore
import AWSS3
```

3. Add the following code to your AppDelegate to establish a run-time connection with AWS Mobile.

```
import UIKit
import AWSCore
import AWSMobileClient

@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate {


    func application(_ application: UIApplication, didFinishLaunchingWithOptions
 launchOptions: [UIApplicationLaunchOptionsKey: Any]?) -> Bool {
        //Instantiate AWSMobileClient to establish AWS user credentials
        return AWSMobileClient.sharedInstance().interceptApplication(application,
 didFinishLaunchingWithOptions: launchOptions)
    }
}
```

Implement Storage Operations

Once your backend is setup and connected to your app, use the following steps to upload and download a file using the SDK's transfer utility.

Upload a File

Android - Java

To upload a file to an Amazon S3 bucket, use `AWSMobileClient` to get the `AWSConfiguration` and `AWSCredentialsProvider`, then create the `TransferUtility` object. `AWSMobileClient` expects an activity context for resuming an authenticated session and creating the credentials provider.

The following example shows using the `TransferUtility `in the context of an Activity. If you are creating :code:`TransferUtility` from an application context, you can construct the `AWSCredentialsProvider` and pass it into `TransferUtility` to use in forming the `AWSConfiguration` object.. The `TransferUtility` will check the size of file being uploaded and will automatically switch over to using multi-part uploads if the file size exceeds 5 MB.

```
import android.app.Activity;
import android.util.Log;

import com.amazonaws.mobile.client.AWSMobileClient;
import com.amazonaws.mobileconnectors.s3.transferutility.TransferUtility;
import com.amazonaws.mobileconnectors.s3.transferutility.TransferState;
import com.amazonaws.mobileconnectors.s3.transferutility.TransferObserver;
import com.amazonaws.mobileconnectors.s3.transferutility.TransferListener;
import com.amazonaws.services.s3.AmazonS3Client;

import java.io.File;

public class YourActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        AWSMobileClient.getInstance().initialize(this).execute();
        uploadWithTransferUtility();
    }

    private void uploadWithTransferUtility() {

        TransferUtility transferUtility =
            TransferUtility.builder()
                .context(getApplicationContext())
                .awsConfiguration(AWSMobileClient.getInstance().getConfiguration())
                .s3Client(new
 AmazonS3Client(AWSMobileClient.getInstance().getCredentialsProvider()))
                .build();

        TransferObserver uploadObserver =
            transferUtility.upload(
                "s3Folder/s3Key.txt",
                new File("/path/to/file/localFile.txt"));

        // Attach a listener to the observer to get state update and progress
 notifications
        uploadObserver.setTransferListener(new TransferListener() {

            @Override
            public void onStateChanged(int id, TransferState state) {
                if (TransferState.COMPLETED == state) {
                    // Handle a completed upload.
```

```
                }
            }

            @Override
            public void onProgressChanged(int id, long bytesCurrent, long bytesTotal) {
                float percentDonef = ((float) bytesCurrent / (float) bytesTotal) * 100;
                int percentDone = (int)percentDonef;

                Log.d("YourActivity", "ID:" + id + " bytesCurrent: " + bytesCurrent
                        + " bytesTotal: " + bytesTotal + " " + percentDone + "%");
            }

            @Override
            public void onError(int id, Exception ex) {
                // Handle errors
            }

        });

        // If you prefer to poll for the data, instead of attaching a
        // listener, check for the state and progress in the observer.
        if (TransferState.COMPLETED == uploadObserver.getState()) {
            // Handle a completed upload.
        }

        Log.d("YourActivity", "Bytes Transferrred: " +
 uploadObserver.getBytesTransferred());
        Log.d("YourActivity", "Bytes Total: " + uploadObserver.getBytesTotal());
    }
}
```

Android - Kotlin

To upload a file to an Amazon S3 bucket, use `AWSMobileClient` to get the `AWSConfiguration` and `AWSCredentialsProvider`, then create the `TransferUtility` object. `AWSMobileClient` expects an activity context for resuming an authenticated session and creating the credentials provider.

The following example shows using the `TransferUtility `in the context of an Activity. If you are creating :code:`TransferUtility` from an application context, you can construct the `AWSCredentialsProvider` and pass it into `TransferUtility` to use in forming the `AWSConfiguration` object.. The `TransferUtility` will check the size of file being uploaded and will automatically switch over to using multi-part uploads if the file size exceeds 5 MB.

```
import android.app.Activity;
import android.util.Log;

import com.amazonaws.mobile.client.AWSMobileClient;
import com.amazonaws.mobileconnectors.s3.transferutility.TransferUtility;
import com.amazonaws.mobileconnectors.s3.transferutility.TransferState;
import com.amazonaws.mobileconnectors.s3.transferutility.TransferObserver;
import com.amazonaws.mobileconnectors.s3.transferutility.TransferListener;
import com.amazonaws.services.s3.AmazonS3Client;

import java.io.File;

class MainActivity : Activity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate()
        AWSMobileClient.getInstance().initialize(this).execute()
        uploadWithTransferUtility(
            "s3Folder/s3Key.txt"
            File("/path/to/file/localfile.txt")
        )
```

```
    }

    private fun uploadWithTransferUtility(remote: String, local: File) {
        val txUtil = TransferUtility.builder()
                .context(getApplicationContext)
                .awsConfiguration(AWSMobileClient.getInstance().configuration)

 .s3Client(AmazonS3Client(AWSMobileClient.getInstance().credentialsProvider))
                .build()

        val txObserver = txUtil.upload(remote, local)
        txObserver.transferListener = object : TransferListener() {
            override fun onStateChanged(id: Int, state: TransferState) {
                if (state == TransferState.COMPLETED) {
                    // Handle a completed upload
                }
            }

            override fun onProgressChanged(id: Int, current: Long, total: Long) {
                val done = (((current / total) * 100.0) as Float) as Int
                Log.d(TAG, "ID: $id, percent done = $done")
            }

            override fun onError(id: Int, ex: Exception) {
                // Handle errors
            }
        }

        // If you prefer to poll for the data, instead of attaching a
        // listener, check for the state and progress in the observer.
        if (txObserver.state == TransferState.COMPLETED) {
            // Handle a completed upload.
        }
    }
}
```

iOS - Swift

The following example shows how to upload a file to an Amazon S3 bucket.

```
func uploadData() {

   let data: Data = Data() // Data to be uploaded

   let expression = AWSS3TransferUtilityUploadExpression()
      expression.progressBlock = {(task, progress) in
         DispatchQueue.main.async(execute: {
           // Do something e.g. Update a progress bar.
        })
   }

   var completionHandler: AWSS3TransferUtilityUploadCompletionHandlerBlock?
   completionHandler = { (task, error) -> Void in
      DispatchQueue.main.async(execute: {
         // Do something e.g. Alert a user for transfer completion.
         // On failed uploads, `error` contains the error object.
      })
   }

   let transferUtility = AWSS3TransferUtility.default()

   transferUtility.uploadData(data,
        bucket: "YourBucket",
        key: "YourFileName",
        contentType: "text/plain",
```

```
        expression: expression,
        completionHandler: completionHandler).continueWith {
           (task) -> AnyObject! in
               if let error = task.error {
                  print("Error: \(error.localizedDescription)")
               }

               if let _ = task.result {
                  // Do something with uploadTask.
               }
               return nil;
       }
}
```

Download a File

Android - Java

To download a file from an Amazon S3 bucket, use `AWSMobileClient` to get the `AWSConfiguration` and `AWSCredentialsProvider` to create the `TransferUtility` object. `AWSMobileClient` expects an activity context for resuming an authenticated session and creating the
<problematic>:cdoe:`AWSCredentialsProvider`</problematic>
.

The following example shows using the `TransferUtility` in the context of an Activity. If you are creating `TransferUtility` from an application context, you can construct the `AWSCredentialsProvider` and pass it into `TransferUtility` to use in forming the `AWSConfiguration` object.

```
import android.app.Activity;
import android.util.Log;

import com.amazonaws.mobile.client.AWSMobileClient;
import com.amazonaws.mobileconnectors.s3.transferutility.TransferUtility;
import com.amazonaws.mobileconnectors.s3.transferutility.TransferState;
import com.amazonaws.mobileconnectors.s3.transferutility.TransferObserver;
import com.amazonaws.mobileconnectors.s3.transferutility.TransferListener;
import com.amazonaws.services.s3.AmazonS3Client;

import java.io.File;

public class YourActivity extends Activity {

    public void dowloadData() {
        AWSMobileClient.getInstance().initialize(this, new AWSStartupHandler() {
            @Override
            public void onComplete() {
                downloadWithTransferUtility();
            }
        }).execute();
    }

    public void downloadWithTransferUtility() {

        TransferUtility transferUtility =
            TransferUtility.builder()
                    .context(getApplicationContext())
                    .awsConfiguration(AWSMobileClient.getInstance().getConfiguration())
                    .s3Client(new
 AmazonS3Client(AWSMobileClient.getInstance().getCredentialsProvider()))
                    .build();
```

```
        TransferObserver downloadObserver =
            transferUtility.download(
                    "s3Folder/s3Key.txt",
                    new File("/path/to/file/localFile.txt"));

        // Attach a listener to the observer to get state update and progress
 notifications
        downloadObserver.setTransferListener(new TransferListener() {

            @Override
            public void onStateChanged(int id, TransferState state) {
                if (TransferState.COMPLETED == state) {
                    // Handle a completed upload.
                }
            }

            @Override
            public void onProgressChanged(int id, long bytesCurrent, long bytesTotal) {
                    float percentDonef = ((float)bytesCurrent/(float)bytesTotal) * 100;
                    int percentDone = (int)percentDonef;

                    Log.d("MainActivity", "   ID:" + id + "   bytesCurrent: " +
 bytesCurrent + "   bytesTotal: " + bytesTotal + " " + percentDone + "%");
            }

            @Override
            public void onError(int id, Exception ex) {
                // Handle errors
            }

        });

        // If you prefer to poll for the data, instead of attaching a
        // listener, check for the state and progress in the observer.
        if (TransferState.COMPLETED == downloadObserver.getState()) {
            // Handle a completed upload.
        }

        Log.d("YourActivity", "Bytes Transferrred: " +
 downloadObserver.getBytesTransferred());
        Log.d("YourActivity", "Bytes Total: " + downloadObserver.getBytesTotal());
    }
}
```

Android - Kotlin

To download a file from an Amazon S3 bucket, use `AWSMobileClient` to get the `AWSConfiguration` and `AWSCredentialsProvider` to create the `TransferUtility` object. `AWSMobileClient` expects an activity context for resuming an authenticated session and creating the
<problematic>:cdoe:`AWSCredentialsProvider`</problematic>
.

The following example shows using the `TransferUtility` in the context of an Activity. If you are creating `TransferUtility` from an application context, you can construct the `AWSCredentialsProvider` and pass it into `TransferUtility` to use in forming the `AWSConfiguration` object.

```
import android.app.Activity;
import android.util.Log;

import com.amazonaws.mobile.client.AWSMobileClient;
import com.amazonaws.mobileconnectors.s3.transferutility.TransferUtility;
```

```
import com.amazonaws.mobileconnectors.s3.transferutility.TransferState;
import com.amazonaws.mobileconnectors.s3.transferutility.TransferObserver;
import com.amazonaws.mobileconnectors.s3.transferutility.TransferListener;
import com.amazonaws.services.s3.AmazonS3Client;

import java.io.File;

class MainActivity : Activity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate()
        AWSMobileClient.getInstance().initialize(this).execute()
        downloadWithTransferUtility(
            "s3Folder/s3Key.txt"
            File("/path/to/file/localfile.txt")
        )
    }

    private fun downloadWithTransferUtility(remote: String, local: File) {
        val txUtil = TransferUtility.builder()
                .context(getApplicationContext)
                .awsConfiguration(AWSMobileClient.getInstance().configuration)

 .s3Client(AmazonS3Client(AWSMobileClient.getInstance().credentialsProvider))
                .build()

        val txObserver = txUtil.download(remote, local)
        txObserver.transferListener = object : TransferListener() {
            override fun onStateChanged(id: Int, state: TransferState) {
                if (state == TransferState.COMPLETED) {
                    // Handle a completed upload
                }
            }

            override fun onProgressChanged(id: Int, current: Long, total: Long) {
                val done = (((current / total) * 100.0) as Float) as Int
                Log.d(TAG, "ID: $id, percent done = $done")
            }

            override fun onError(id: Int, ex: Exception) {
                // Handle errors
            }
        }

        // If you prefer to poll for the data, instead of attaching a
        // listener, check for the state and progress in the observer.
        if (txObserver.state == TransferState.COMPLETED) {
            // Handle a completed upload.
        }
    }
}
```

iOS - Swift

The following example shows how to download a file from an Amazon S3 bucket.

```
func downloadData() {
   let expression = AWSS3TransferUtilityDownloadExpression()
   expression.progressBlock = {(task, progress) in DispatchQueue.main.async(execute: {
      // Do something e.g. Update a progress bar.
      })
   }

   var completionHandler: AWSS3TransferUtilityDownloadCompletionHandlerBlock?
   completionHandler = { (task, URL, data, error) -> Void in
      DispatchQueue.main.async(execute: {
```

```
      // Do something e.g. Alert a user for transfer completion.
      // On failed downloads, `error` contains the error object.
      })
   }

   let transferUtility = AWSS3TransferUtility.default()
   transferUtility.downloadData(
         fromBucket: "YourBucket",
         key: "YourFileName",
         expression: expression,
         completionHandler: completionHandler
         ).continueWith {
            (task) -> AnyObject! in if let error = task.error {
               print("Error: \(error.localizedDescription)")
            }

            if let _ = task.result {
              // Do something with downloadTask.

            }
            return nil;
        }
}
```

Next Steps

- For further information about TransferUtility capabilities, see Transfer Files and Data Using TransferUtility and Amazon S3 (p. 194).
- For sample apps that demonstrate TransferUtility capabilities, see Android S3 TransferUtility Sample and iOS S3 TransferUtility Sample.

Transfer Files and Data Using TransferUtility and Amazon S3

Just Getting Started?	Use streamlined steps (p. 66) to install the SDK and integrate Amazon S3.

Or, use the contents of this page if your app will integrate existing AWS services.

This page explains how to implement upload and download functionality and a number of additional storage use cases.

The examples on this page assume you have added the the AWS Mobile SDK to your mobile app. To create a new cloud storage backend for your app, see Add User File Storage (p. 66).

Best practice	If you use the transfer utility multipart upload feature, take advantage of automatic cleanup features by seting up the AbortIncompleteMultipartUpload action in your Amazon S3 bucket life cycle configuration.

Upload a File

Android - Java

The following example shows how to upload a file to an Amazon S3 bucket.

Use `AWSMobileClient` to get the `AWSConfiguration` and `AWSCredentialsProvider`, then create the `TransferUtility` object. AWSMobileClient expects an activity context for resuming an authenticated session and creating the credentials provider.

The following example shows using the transfer utility in the context of an Activity. If you are creating transfer utility from an application context, you can construct the CredentialsProvider and AWSConfiguration object and pass it into TransferUtility. The TransferUtility will check the size of file being uploaded and will automatically switch over to using multi part uploads if the file size exceeds 5 MB.

```
import android.app.Activity;
import android.util.Log;

import com.amazonaws.mobile.client.AWSMobileClient;
import com.amazonaws.mobileconnectors.s3.transferutility.TransferUtility;
import com.amazonaws.mobileconnectors.s3.transferutility.TransferState;
import com.amazonaws.mobileconnectors.s3.transferutility.TransferObserver;
import com.amazonaws.mobileconnectors.s3.transferutility.TransferListener;
import com.amazonaws.services.s3.AmazonS3Client;

import java.io.File;

public class YourActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        AWSMobileClient.getInstance().initialize(this).execute();
        uploadWithTransferUtility();
    }

    private void uploadWithTransferUtility() {

        TransferUtility transferUtility =
            TransferUtility.builder()
                .context(getApplicationContext())
                .awsConfiguration(AWSMobileClient.getInstance().getConfiguration())
                .s3Client(new
 AmazonS3Client(AWSMobileClient.getInstance().getCredentialsProvider()))
                .build();

        TransferObserver uploadObserver =
            transferUtility.upload(
                "s3Folder/s3Key.txt",
                new File("/path/to/file/localFile.txt"));

        // Attach a listener to the observer to get state update and progress
 notifications
        uploadObserver.setTransferListener(new TransferListener() {

            @Override
            public void onStateChanged(int id, TransferState state) {
                if (TransferState.COMPLETED == state) {
                    // Handle a completed upload.
                }
            }

            @Override
            public void onProgressChanged(int id, long bytesCurrent, long bytesTotal) {
                float percentDonef = ((float) bytesCurrent / (float) bytesTotal) * 100;
                int percentDone = (int)percentDonef;

                Log.d("YourActivity", "ID:" + id + " bytesCurrent: " + bytesCurrent
                        + " bytesTotal: " + bytesTotal + " " + percentDone + "%");
            }
```

```
            @Override
            public void onError(int id, Exception ex) {
                // Handle errors
            }

        });

        // If you prefer to poll for the data, instead of attaching a
        // listener, check for the state and progress in the observer.
        if (TransferState.COMPLETED == uploadObserver.getState()) {
            // Handle a completed upload.
        }

        Log.d("YourActivity", "Bytes Transferrred: " +
 uploadObserver.getBytesTransferred());
        Log.d("YourActivity", "Bytes Total: " + uploadObserver.getBytesTotal());
    }
}
```

Android - Kotlin

The following example shows how to upload a file to an Amazon S3 bucket.

Use `AWSMobileClient` to get the `AWSConfiguration` and `AWSCredentialsProvider`, then create the `TransferUtility` object. AWSMobileClient expects an activity context for resuming an authenticated session and creating the credentials provider.

The following example shows using the transfer utility in the context of an Activity. If you are creating transfer utility from an application context, you can construct the CredentialsProvider and AWSConfiguration object and pass it into TransferUtility. The TransferUtility will check the size of file being uploaded and will automatically switch over to using multi part uploads if the file size exceeds 5 MB.

```
import android.app.Activity;
import android.util.Log;

import com.amazonaws.mobile.client.AWSMobileClient;
import com.amazonaws.mobileconnectors.s3.transferutility.TransferUtility;
import com.amazonaws.mobileconnectors.s3.transferutility.TransferState;
import com.amazonaws.mobileconnectors.s3.transferutility.TransferObserver;
import com.amazonaws.mobileconnectors.s3.transferutility.TransferListener;
import com.amazonaws.services.s3.AmazonS3Client;

import java.io.File;

class MainActivity : Activity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate()
        AWSMobileClient.getInstance().initialize(this).execute()
        uploadWithTransferUtility(
            "s3Folder/s3Key.txt"
            File("/path/to/file/localfile.txt")
        )
    }

    private fun uploadWithTransferUtility(remote: String, local: File) {
        val txUtil = TransferUtility.builder()
                .context(getApplicationContext)
                .awsConfiguration(AWSMobileClient.getInstance().configuration)

 .s3Client(AmazonS3Client(AWSMobileClient.getInstance().credentialsProvider))
                .build()
```

```
        val txObserver = txUtil.upload(remote, local)
        txObserver.transferListener = object : TransferListener() {
            override fun onStateChanged(id: Int, state: TransferState) {
                if (state == TransferState.COMPLETED) {
                    // Handle a completed upload
                }
            }

            override fun onProgressChanged(id: Int, current: Long, total: Long) {
                val done = (((current / total) * 100.0) as Float) as Int
                Log.d(TAG, "ID: $id, percent done = $done")
            }

            override fun onError(id: Int, ex: Exception) {
                // Handle errors
            }
        }

        // If you prefer to poll for the data, instead of attaching a
        // listener, check for the state and progress in the observer.
        if (txObserver.state == TransferState.COMPLETED) {
            // Handle a completed upload.
        }
    }
}
```

iOS - Swift

The transfer utility provides methods for both single-part and multipart uploads. When a transfer uses multipart upload, the data is chunked into a number of 5 MB parts which are transferred in parallel for increased speed.

The following example shows how to upload a file to an Amazon S3 bucket.

```
func uploadData() {

   let data: Data = Data() // Data to be uploaded

   let expression = AWSS3TransferUtilityUploadExpression()
      expression.progressBlock = {(task, progress) in
         DispatchQueue.main.async(execute: {
           // Do something e.g. Update a progress bar.
         })
   }

   var completionHandler: AWSS3TransferUtilityUploadCompletionHandlerBlock?
   completionHandler = { (task, error) -> Void in
      DispatchQueue.main.async(execute: {
         // Do something e.g. Alert a user for transfer completion.
         // On failed uploads, `error` contains the error object.
      })
   }

   let transferUtility = AWSS3TransferUtility.default()

   transferUtility.uploadData(data,
        bucket: "YourBucket",
        key: "YourFileName",
        contentType: "text/plain",
        expression: expression,
        completionHandler: completionHandler).continueWith {
           (task) -> AnyObject! in
               if let error = task.error {
                  print("Error: \(error.localizedDescription)")
```

```
                }

                if let _ = task.result {
                   // Do something with uploadTask.
                }
                return nil;
        }
}
```

The following example shows how to upload a file to an Amazon S3 bucket using multipart uploads.

```
func uploadData() {

   let data: Data = Data() // Data to be uploaded

   let expression = AWSS3TransferUtilityMultiPartUploadExpression()
      expression.progressBlock = {(task, progress) in
         DispatchQueue.main.async(execute: {
           // Do something e.g. Update a progress bar.
       })
   }

   var completionHandler: AWSS3TransferUtilityMultiPartUploadCompletionHandlerBlock
   completionHandler = { (task, error) -> Void in
      DispatchQueue.main.async(execute: {
         // Do something e.g. Alert a user for transfer completion.
         // On failed uploads, `error` contains the error object.
      })
   }

   let transferUtility = AWSS3TransferUtility.default()

   transferUtility.uploadUsingMultiPart(data:data,
        bucket: "YourBucket",
        key: "YourFileName",
        contentType: "text/plain",
        expression: expression,
        completionHandler: completionHandler).continueWith {
           (task) -> AnyObject! in
               if let error = task.error {
                  print("Error: \(error.localizedDescription)")
               }

               if let _ = task.result {
                  // Do something with uploadTask.
               }
               return nil;
        }
}
```

Download a File

Android - Java

The following example shows how to download a file from an Amazon S3 bucket. We use `AWSMobileClient` to get the `AWSConfiguration` and `AWSCredentialsProvider` to create the `TransferUtility` object. AWSMobileClient expects an activity context for resuming an authenticated session and creating the credentials provider.

This example shows using the transfer utility in the context of an Activity. If you are creating transfer utility from an application context, you can construct the CredentialsProvider and AWSConfiguration object and pass it into TransferUtility.

```
import android.app.Activity;
import android.util.Log;

import com.amazonaws.mobile.client.AWSMobileClient;
import com.amazonaws.mobileconnectors.s3.transferutility.TransferUtility;
import com.amazonaws.mobileconnectors.s3.transferutility.TransferState;
import com.amazonaws.mobileconnectors.s3.transferutility.TransferObserver;
import com.amazonaws.mobileconnectors.s3.transferutility.TransferListener;
import com.amazonaws.services.s3.AmazonS3Client;

import java.io.File;

public class YourActivity extends Activity {

    @Override
      protected void onCreate(Bundle savedInstanceState) {
          AWSMobileClient.getInstance().initialize(this).execute();
          downloadWithTransferUtility();
    }

    public void downloadWithTransferUtility() {

        TransferUtility transferUtility =
              TransferUtility.builder()
                    .context(getApplicationContext())
                    .awsConfiguration(AWSMobileClient.getInstance().getConfiguration())
                    .s3Client(new
 AmazonS3Client(AWSMobileClient.getInstance().getCredentialsProvider()))
                    .build();

        TransferObserver downloadObserver =
              transferUtility.download(
                    "s3Folder/s3Key.txt",
                    new File("/path/to/file/localFile.txt"));

        // Attach a listener to the observer to get notified of the
        // updates in the state and the progress
        downloadObserver.setTransferListener(new TransferListener() {

           @Override
           public void onStateChanged(int id, TransferState state) {
              if (TransferState.COMPLETED == state) {
                 // Handle a completed upload.
              }
           }

           @Override
           public void onProgressChanged(int id, long bytesCurrent, long bytesTotal) {
                 float percentDonef = ((float)bytesCurrent/(float)bytesTotal) * 100;
                 int percentDone = (int)percentDonef;

                 Log.d("MainActivity", "   ID:" + id + "   bytesCurrent: " +
 bytesCurrent + "   bytesTotal: " + bytesTotal + " " + percentDone + "%");
           }

           @Override
           public void onError(int id, Exception ex) {
              // Handle errors
           }

        });

        // If you do not want to attach a listener and poll for the data
        // from the observer, you can check for the state and the progress
        // in the observer.
```

```
        if (TransferState.COMPLETED == downloadObserver.getState()) {
            // Handle a completed upload.
        }

        Log.d("YourActivity", "Bytes Transferrred: " +
 downloadObserver.getBytesTransferred());
        Log.d("YourActivity", "Bytes Total: " + downloadObserver.getBytesTotal());
    }
}
```

Android - Kotlin

The following example shows how to download a file from an Amazon S3 bucket. We use `AWSMobileClient` to get the `AWSConfiguration` and `AWSCredentialsProvider` to create the `TransferUtility` object. AWSMobileClient expects an activity context for resuming an authenticated session and creating the credentials provider.

This example shows using the transfer utility in the context of an Activity. If you are creating transfer utility from an application context, you can construct the CredentialsProvider and AWSConfiguration object and pass it into TransferUtility.

```
import android.app.Activity;
import android.util.Log;

import com.amazonaws.mobile.client.AWSMobileClient;
import com.amazonaws.mobileconnectors.s3.transferutility.TransferUtility;
import com.amazonaws.mobileconnectors.s3.transferutility.TransferState;
import com.amazonaws.mobileconnectors.s3.transferutility.TransferObserver;
import com.amazonaws.mobileconnectors.s3.transferutility.TransferListener;
import com.amazonaws.services.s3.AmazonS3Client;

import java.io.File;

class MainActivity : Activity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate()
        AWSMobileClient.getInstance().initialize(this).execute()
        downloadWithTransferUtility(
            "s3Folder/s3Key.txt"
            File("/path/to/file/localfile.txt")
        )
    }

    private fun downloadWithTransferUtility(remote: String, local: File) {
        val txUtil = TransferUtility.builder()
                .context(getApplicationContext)
                .awsConfiguration(AWSMobileClient.getInstance().configuration)

 .s3Client(AmazonS3Client(AWSMobileClient.getInstance().credentialsProvider))
                .build()

        val txObserver = txUtil.download(remote, local)
        txObserver.transferListener = object : TransferListener() {
            override fun onStateChanged(id: Int, state: TransferState) {
                if (state == TransferState.COMPLETED) {
                    // Handle a completed upload
                }
            }

            override fun onProgressChanged(id: Int, current: Long, total: Long) {
                val done = (((current / total) * 100.0) as Float) as Int
                Log.d(TAG, "ID: $id, percent done = $done")
            }
```

```
            override fun onError(id: Int, ex: Exception) {
                // Handle errors
            }
        }

        // If you prefer to poll for the data, instead of attaching a
        // listener, check for the state and progress in the observer.
        if (txObserver.state == TransferState.COMPLETED) {
            // Handle a completed upload.
        }
    }
}
```

iOS - Swift

The following example shows how to download a file from an Amazon S3 bucket.

```
func downloadData() {
   let expression = AWSS3TransferUtilityDownloadExpression()
   expression.progressBlock = {(task, progress) in DispatchQueue.main.async(execute: {
      // Do something e.g. Update a progress bar.
      })
   }

   var completionHandler: AWSS3TransferUtilityDownloadCompletionHandlerBlock?
   completionHandler = { (task, URL, data, error) -> Void in
      DispatchQueue.main.async(execute: {
      // Do something e.g. Alert a user for transfer completion.
      // On failed downloads, `error` contains the error object.
      })
   }

   let transferUtility = AWSS3TransferUtility.default()
   transferUtility.downloadData(
         fromBucket: "YourBucket",
         key: "YourFileName",
         expression: expression,
         completionHandler: completionHandler
         ).continueWith {
            (task) -> AnyObject! in if let error = task.error {
               print("Error: \(error.localizedDescription)")
            }

            if let _ = task.result {
              // Do something with downloadTask.

            }
            return nil;
        }
}
```

Track Transfer Progress

Android - Java

With the `TransferUtility`, the download() and upload() methods return a `TransferObserver` object. This object gives access to:

1. The state, as an `enum`
2. The total bytes currently transferred
3. The total bytes remaining to transfer, to aid in calculating progress bars

4. A unique ID that you can use to keep track of distinct transfers

Given the transfer ID, the `TransferObserver` object can be retrieved from anywhere in your app, even if the app was terminated during a transfer. It also lets you create a `TransferListener`, which will be updated on state or progress change, as well as when an error occurs.

To get the progress of a transfer, call `setTransferListener()` on your `TransferObserver`. This requires you to implement `onStateChanged`, `onProgressChanged`, and `onError`. For example:

You can also query for `TransferObservers` with either the `getTransfersWithType(transferType)` or `getTransfersWithTypeAndState(transferType, transferState)` method. You can use `TransferObservers` to determine what transfers are underway, what are paused and handle the transfers as necessary.

```
TransferObserver transferObserver = download(MY_BUCKET, OBJECT_KEY, MY_FILE);
transferObserver.setTransferListener(new TransferListener(){

    @Override
    public void onStateChanged(int id, TransferState state) {
        // do something
    }

    @Override
    public void onProgressChanged(int id, long bytesCurrent, long bytesTotal) {
        int percentage = (int) (bytesCurrent/bytesTotal * 100);
        //Display percentage transfered to user
    }

    @Override
    public void onError(int id, Exception ex) {
        // do something
    }
});
```

The transfer ID can be retrieved from the `TransferObserver` object that is returned from upload or download function.

```
// Gets id of the transfer.
int transferId = transferObserver.getId();
```

Android - Kotlin

With the `TransferUtility`, the download() and upload() methods return a `TransferObserver` object. This object gives access to:

1. The state, as an `enum`
2. The total bytes currently transferred
3. The total bytes remaining to transfer, to aid in calculating progress bars
4. A unique ID that you can use to keep track of distinct transfers

Given the transfer ID, the `TransferObserver` object can be retrieved from anywhere in your app, even if the app was terminated during a transfer. It also lets you create a `TransferListener`, which will be updated on state or progress change, as well as when an error occurs.

To get the progress of a transfer, call `setTransferListener()` on your `TransferObserver`. This requires you to implement `onStateChanged`, `onProgressChanged`, and `onError`. For example:

You can also query for `TransferObservers` with either the `getTransfersWithType(transferType)` or `getTransfersWithTypeAndState(transferType, transferState)` method. You can use `TransferObservers` to determine what transfers are underway, what are paused and handle the transfers as necessary.

```
val transferObserver = download(MY_BUCKET, OBJECT_KEY, MY_FILE);
transferObserver.transferListener = object : TransferListener() {
    override fun onStateChanged(id: Int, state: TransferState) {
        // Do something
    }

    override fun onProgressChanged(id: int, current: Long, total: Long) {
        int percent = ((current / total) * 100.0) as Int
        // Display percent transferred
    }

    override fun onError(id: Int, ex: Exception) {
        // Do something
    }
}
```

The transfer ID can be retrieved from the `TransferObserver` object that is returned from upload or download function.

```
// Gets id of the transfer.
val transferId = transferObserver.id;
```

iOS - Swift

Implement progress and completion actions for transfers by passing a `progressBlock` and `completionHandler` blocks to the call to `AWSS3TransferUtility` that initiates the transfer.

The following example of initiating a data upload, shows how progress and completion handling is typically done for all transfers. The `AWSS3TransferUtilityUploadExpression`, `AWSS3TransferUtilityMultiPartUploadExpression` and `AWSS3TransferUtilityDownloadExpression` contains the `progressBlock` that gives you the progress of the transfer which you can use to update the progress bar.

```
// For example, create a progress bar
let progressView: UIProgressView! = UIProgressView()
progressView.progress = 0.0;

let data = Data() // The data to upload

let expression = AWSS3TransferUtilityUploadExpression()
expression.progressBlock = {(task, progress) in DispatchQueue.main.async(execute: {
        // Update a progress bar.
        progressView.progress = Float(progress.fractionCompleted)
    })
}

let completionHandler: AWSS3TransferUtilityUploadCompletionHandlerBlock = { (task,
 error) -> Void in DispatchQueue.main.async(execute: {
        if let error = error {
            NSLog("Failed with error: \(error)")
        }
        else if(self.progressView.progress != 1.0) {
            NSLog("Error: Failed.")
        } else {
            NSLog("Success.")
```

```
        }
    })
}

var refUploadTask: AWSS3TransferUtilityTask?
let transferUtility = AWSS3TransferUtility.default()
transferUtility.uploadData(data,
           bucket: "S3BucketName",
           key: "S3UploadKeyName",
           contentType: "text/plain",
           expression: expression,
           completionHandler: completionHandler).continueWith { (task) -> AnyObject! in
                if let error = task.error {
                    print("Error: \(error.localizedDescription)")
                }

                if let uploadTask = task.result {
                    // Do something with uploadTask.
                    // The uploadTask can be used to pause/resume/cancel the operation,
 retrieve task specific information
                    refUploadTask = uploadTask
                }

                return nil;
            }
```

Pause a Transfer

Android - Java

Transfers can be paused using the `pause(transferId)` method. If your app is terminated, crashes, or loses Internet connectivity, transfers are automatically paused.

The `transferId` can be retrieved from the `TransferObserver` object as described in Track Transfer Progress (p. 201).

To pause a single transfer:

```
transferUtility.pause(idOfTransferToBePaused);
```

To pause all uploads:

```
transferUtility.pauseAllWithType(TransferType.UPLOAD);
```

To pause all downloads:

```
transferUtility.pauseAllWithType(TransferType.DOWNLOAD);
```

To pause all transfers of any type:

```
transferUtility.pauseAllWithType(TransferType.ANY);
```

Android - Kotlin

Transfers can be paused using the `pause(transferId)` method. If your app is terminated, crashes, or loses Internet connectivity, transfers are automatically paused.

The `transferId` can be retrieved from the `TransferObserver` object as described in Track Transfer Progress (p. 201).

To pause a single transfer:

```
transferUtility.pause(idOfTransferToBePaused);
```

To pause all uploads:

```
transferUtility.pauseAllWithType(TransferType.UPLOAD);
```

To pause all downloads:

```
transferUtility.pauseAllWithType(TransferType.DOWNLOAD);
```

To pause all transfers of any type:

```
transferUtility.pauseAllWithType(TransferType.ANY);
```

iOS - Swift

To pause or suspend a transfer, retain references to `AWSS3TransferUtilityUploadTask`, `AWSS3TransferUtilityMultiPartUploadTask` or `AWSS3TransferUtilityDownloadTask` .

As described in the previous section Track Transfer Progress (p. 201), the variable `refUploadTask` is a reference to the `UploadTask` object that is retrieved from the `continueWith` block of an upload operation that is invoked through `transferUtility.uploadData`.

To pause a transfer, use the `suspend` method:

```
refUploadTask.suspend()
```

Resume a Transfer

Android - Java

In the case of a loss in network connectivity, transfers will automatically resume when network connectivity is restored. If the app crashed or was terminated by the operating system, transfers can be resumed with the `resume(transferId)` method.

The `transferId` can be retrieved from the `TransferObserver` object as described in Track Transfer Progress (p. 201).

To resume a single transfer:

```
transferUtility.resume(idOfTransferToBeResumed);
```

To resume all uploads:

```
transferUtility.resumeAllWithType(TransferType.UPLOAD);
```

To resume all downloads:

```
transferUtility.resumeAllWithType(TransferType.DOWNLOAD);
```

To resume all transfers of any type:

```
transferUtility.resumeAllWithType(TransferType.ANY);
```

Android - Kotlin

In the case of a loss in network connectivity, transfers will automatically resume when network connectivity is restored. If the app crashed or was terminated by the operating system, transfers can be resumed with the `resume(transferId)` method.

The `transferId` can be retrieved from the `TransferObserver` object as described in Track Transfer Progress (p. 201).

To resume a single transfer:

```
transferUtility.resume(idOfTransferToBeResumed);
```

To resume all uploads:

```
transferUtility.resumeAllWithType(TransferType.UPLOAD);
```

To resume all downloads:

```
transferUtility.resumeAllWithType(TransferType.DOWNLOAD);
```

To resume all transfers of any type:

```
transferUtility.resumeAllWithType(TransferType.ANY);
```

iOS - Swift

To resume an upload or a download operation, retain references to `AWSS3TransferUtilityUploadTask`, `AWSS3TransferUtilityMultiPartUploadTask` or `AWSS3TransferUtilityDownloadTask`.

As described in the previous section Track Transfer Progress (p. 201), the variable `refUploadTask` is a reference to the `UploadTask` object that is retrieved from the `continueWith` block of an upload operation that is invoked through `transferUtility.uploadData`.

To resume a transfer, use the `resume` method:

```
refUploadTask.resume()
```

Cancel a Transfer

Android - Java

To cancel an upload, call cancel() or cancelAllWithType() on the `TransferUtility` object.

The `transferId` can be retrieved from the `TransferObserver` object as described in Track Transfer Progress (p. 201).

To cancel a single transfer, use:

```
transferUtility.cancel(idToBeCancelled);
```

To cancel all transfers of a certain type, use:

```
transferUtility.cancelAllWithType(TransferType.DOWNLOAD);
```

Android - Kotlin

To cancel an upload, call cancel() or cancelAllWithType() on the `TransferUtility` object.

The `transferId` can be retrieved from the `TransferObserver` object as described in Track Transfer Progress (p. 201).

To cancel a single transfer, use:

```
transferUtility.cancel(idToBeCancelled);
```

To cancel all transfers of a certain type, use:

```
transferUtility.cancelAllWithType(TransferType.DOWNLOAD);
```

iOS - Swift

To cancel an upload or a download operation, retain references to `AWSS3TransferUtilityUploadTask`, `AWSS3TransferUtilityMultiPartUploadTask` and `AWSS3TransferUtilityDownloadTask`.

As described in the previous section Track Transfer Progress (p. 201), the variable `refUploadTask` is a reference to the `UploadTask` object that is retrieved from the `continueWith` block of an upload operation that is invoked through `transferUtility.uploadData`.

To cancel a transfer, use the `cancel` method:

```
refUploadTask.cancel()
```

Background Transfers

The SDK supports uploading to and downloading from Amazon S3 while your app is running in the background.

Android - Java

No additional work is needed to use this feature. As long as your app is present in the background a transfer that is in progress will continue.

Android - Kotlin

No additional work is needed to use this feature. As long as your app is present in the background a transfer that is in progress will continue.

iOS - Swift

Configure the Application Delegate

The `TransferUtility` for iOS uses NSURLSession background transfers to continue data transfers even when your app moves to the background. Call the following method in the – `application:handleEventsForBackgroundURLSession: completionHandler:` of your

application delegate. When the app moves the foreground, the delegate enables iOS to notify TransferUtility that a transfer has completed.

```
func application(_ application: UIApplication, handleEventsForBackgroundURLSession
 identifier: String, completionHandler: @escaping () -> Void) {
    // Store the completion handler.
    AWSS3TransferUtility.interceptApplication(application,
 handleEventsForBackgroundURLSession: identifier, completionHandler: completionHandler)
}
```

Manage a Transfer with the App in the Foreground

To manage transfers for an app that has moved from the background to the foregroud, retain references to `AWSS3TransferUtilityUploadTask`, `AWSS3TransferUtilityMultiPartUploadTask` and `AWSS3TransferUtilityDownloadTask`. Call suspend, resume, or cancel methods on those task references. The following example shows how to suspend a transfer when the app is about to be terminated.

```
transferUtility.uploadFile(fileURL,
        bucket: S3BucketName,
        key: S3UploadKeyName,
        contentType: "image/png",
        expression: nil,
        completionHandler: nil).continueWith {
            (task) -> AnyObject! in if let error = task.error {
                print("Error: \(error.localizedDescription)")
            }

            if let uploadTask = task.result {
                uploadTask.suspend()
            }

            return nil;
        }
```

Manage a Transfer when a Suspended App Returns to the Foreground

When an app that has initiated a transfer becomes suspended and then returns to the foreground, the transfer may still be in progress or may have completed. In both cases, use the following code to reestablish the progress and completion handler blocks of the app.

This code example is for downloading a file but the same pattern can be used for upload:

You can get a reference to the `AWSS3TransferUtilityUploadTask`, `AWSS3TransferUtilityMultiPartUploadTask` and `AWSS3TransferUtilityDownloadTask` objects from the task.result in continueWith block when you initiate the upload and download respectively. These tasks have a property called taskIdentifier, which uniquely identifies the transfer task object within the `AWSS3TransferUtility`. Your app should persist the identifier through closure and relaunch, so that you can uniquely identify the task objects when the app comes back into the foreground.

```
let transferUtility = AWSS3TransferUtility.default()

var uploadProgressBlock: AWSS3TransferUtilityProgressBlock? = {(task:
 AWSS3TransferUtilityTask, progress: Progress) in
    DispatchQueue.main.async {
        // Handle progress feedback, e.g. update progress bar
    }
}


var downloadProgressBlock: AWSS3TransferUtilityProgressBlock? = {
```

```
    (task: AWSS3TransferUtilityTask, progress: Progress) in DispatchQueue.main.async {
        // Handle progress feedback, e.g. update progress bar
    }
}
var completionBlockUpload:AWSS3TransferUtilityUploadCompletionHandlerBlock? = {
    (task, error) in DispatchQueue.main.async {
        // perform some action on completed upload operation
    }
}
var completionBlockDownload:AWSS3TransferUtilityDownloadCompletionHandlerBlock? = {
    (task, url, data, error) in DispatchQueue.main.async {
        // perform some action on completed download operation
    }
}



transferUtility.enumerateToAssignBlocks(forUploadTask: {
    (task, progress, completion) -> Void in

        let progressPointer =
 AutoreleasingUnsafeMutablePointer<AWSS3TransferUtilityProgressBlock?>(&
 uploadProgressBlock)

        let completionPointer =
 AutoreleasingUnsafeMutablePointer<AWSS3TransferUtilityUploadCompletionHandlerBlock?
>(&completionBlockUpload)

        // Reassign your progress feedback
        progress?.pointee = progressPointer.pointee

        // Reassign your completion handler.
        completion?.pointee = completionPointer.pointee

}, downloadTask: {
    (task, progress, completion) -> Void in

        let progressPointer =
 AutoreleasingUnsafeMutablePointer<AWSS3TransferUtilityProgressBlock?
>(&downloadProgressBlock)

        let completionPointer =
 AutoreleasingUnsafeMutablePointer<AWSS3TransferUtilityDownloadCompletionHandlerBlock?
>(&completionBlockDownload)

        // Reassign your progress feedback
        progress?.pointee = progressPointer.pointee

        // Reassign your completion handler.
        completion?.pointee = completionPointer.pointee
})

 if let downloadTask = task.result {
    // Do something with downloadTask.
}
```

Advanced Transfer Methods

Topics

Transfer with Object Metadata

Android - Java

To upload a file with metadata, use the `ObjectMetadata` object. Create a `ObjectMetadata` object and add in the metadata headers and pass it to the upload function.

```
import com.amazonaws.services.s3.model.ObjectMetadata;

ObjectMetadata myObjectMetadata = new ObjectMetadata();

//create a map to store user metadata
Map<String, String> userMetadata = new HashMap<String,String>();
userMetadata.put("myKey","myVal");

//call setUserMetadata on our ObjectMetadata object, passing it our map
myObjectMetadata.setUserMetadata(userMetadata);
```

Then, upload an object along with its metadata:

```
TransferObserver observer = transferUtility.upload(
  MY_BUCKET,        /* The bucket to upload to */
  OBJECT_KEY,       /* The key for the uploaded object */
  MY_FILE,          /* The file where the data to upload exists */
  myObjectMetadata  /* The ObjectMetadata associated with the object*/
);
```

To download the meta, use the S3 `getObjectMetadata` method. For more information, see the API Reference.

Android - Kotlin

To upload a file with metadata, use the `ObjectMetadata` object. Create a `ObjectMetadata` object and add in the metadata headers and pass it to the upload function.

```
import com.amazonaws.services.s3.model.ObjectMetadata;

val myObjectMetadata = new ObjectMetadata()
myObjectMetadata.userMetadata = mapOf("myKey" to "myVal")
```

Then, upload an object along with its metadata:

```
val observer = transferUtility.upload(
  MY_BUCKET,        /* The bucket to upload to */
  OBJECT_KEY,       /* The key for the uploaded object */
  MY_FILE,          /* The file where the data to upload exists */
  myObjectMetadata  /* The ObjectMetadata associated with the object*/
)
```

To download the meta, use the S3 `getObjectMetadata` method. For more information, see the API Reference.

iOS - Swift

`AWSS3TransferUtilityUploadExpression` and `AWSS3TransferUtilityMultiPartUploadExpression` contain the method *setValue:forRequestHeader* where you can pass in metadata to Amazon S3. This example demonstrates passing in the Server-side Encryption Algorithm as a request header in uploading data to S3 using MultiPart.

```
let data: Data = Data() // The data to upload

let uploadExpression = AWSS3TransferUtilityMultiPartUploadExpression()
uploadExpression.setValue("AES256", forRequestHeader: "x-amz-server-side-encryption-
customer-algorithm")
uploadExpression.progressBlock = {(task, progress) in DispatchQueue.main.async(execute:
 {
        // Do something e.g. Update a progress bar.
    })
}

let transferUtility = AWSS3TransferUtility.default()

transferUtility.uploadUsingMultiPart(data:data,
            bucket: "S3BucketName",
            key: "S3UploadKeyName",
            contentType: "text/plain",
            expression: uploadExpression,
            completionHandler: nil).continueWith { (task) -> AnyObject! in
                if let error = task.error {
                    print("Error: \(error.localizedDescription)")
                }

                return nil;
            }
```

Transfer with Access Control List

Android - Java

To upload a file with Access Control List, use the `CannedAccessControlList` object. The CannedAccessControlList specifies the constants defining a canned access control list. For example, if you use CannedAccessControlList.PublicRead , this specifies the owner is granted `Permission.FullControl` and the `GroupGrantee.AllUsers` group grantee is granted Permission.Read access.

Then, upload an object along with its ACL:

```
TransferObserver observer = transferUtility.upload(
  MY_BUCKET,                        /* The bucket to upload to */
  OBJECT_KEY,                       /* The key for the uploaded object */
  MY_FILE,                          /* The file where the data to upload exists */
  CannedAccessControlList.PublicRead  /* Specify PublicRead ACL for the object in the
 bucket. */
);
```

Android - Kotlin

To upload a file with Access Control List, use the `CannedAccessControlList` object. The CannedAccessControlList specifies the constants defining a canned access control list. For example, if you use CannedAccessControlList.PublicRead , this specifies the owner is granted `Permission.FullControl` and the `GroupGrantee.AllUsers` group grantee is granted Permission.Read access.

Then, upload an object along with its ACL:

```
val observer = transferUtility.upload(
  MY_BUCKET,                        /* The bucket to upload to */
  OBJECT_KEY,                       /* The key for the uploaded object */
  MY_FILE,                          /* The file where the data to upload exists */
```

```
  CannedAccessControlList.PublicRead  /* Specify PublicRead ACL for the object in the
 bucket. */
)
```

iOS - Swift

To upload a file and specify permissions for it, you can use predefined grants, also known as canned ACLs. The following code shows you how to setup a file with publicRead access using the AWSS3 client.

```
//Create a AWSS3PutObjectRequest object and setup the content, bucketname, key on it.
//use the .acl method to specify the ACL for the file
let s3: AWSS3 = AWSS3.default()

let putObjectRequest: AWSS3PutObjectRequest! = AWSS3PutObjectRequest()
let content = "testObjectData"
putObjectRequest.acl = AWSS3ObjectCannedACL.publicRead
putObjectRequest.bucket = "bucket name"
putObjectRequest.key = "file name"
putObjectRequest.body = content
putObjectRequest.contentLength = content.count as NSNumber
putObjectRequest.contentType = "text/plain";

s3.putObject(putObjectRequest, completionHandler:
 { (putObjectOutput:AWSS3PutObjectOutput? , error: Error? ) in
    if let output = putObjectOutput {
        print (output)
    }

    if let error = error {
        print (error)
    }
})
```

Transfer Utility Options

Android - Java

You can use the `TransferUtilityOptions` object to customize the operations of the `TransferUtility`.

TransferThreadPoolSize This parameter will let you specify the number of threads in the thread pool for transfers. By increasing the number of threads, you will be able to increase the number of parts of a mulit-part upload that will be uploaded in parallel. By default, this is set to 2 * (N + 1), where N is the number of available processors on the mobile device. The minimum allowed value is 2.

```
TransferUtilityOptions options = new TransferUtilityOptions();
options.setTransferThreadPoolSize(8);

TransferUtility transferUtility = TransferUtility.builder()
    // Pass-in S3Client, Context, AWSConfiguration/DefaultBucket Name
    .transferUtilityOptions(options)
    .build();
```

TransferServiceCheckTimeInterval The `TransferUtility` monitors each on-going transfer by checking its status periodically. If a stalled transfer is detected, it will be automatically resumed by the `TransferUtility`. The TransferServiceCheckTimeInterval option allows you to set the time interval between the status checks. It is specified in milliseconds and set to 60,000 by default.

```
TransferUtilityOptions options = new TransferUtilityOptions();
options.setTransferServiceCheckTimeInterval(2 * 60 * 1000); // 2-minutes

TransferUtility transferUtility = TransferUtility.builder()
    // Pass-in S3Client, Context, AWSConfiguration/DefaultBucket Name
    .transferUtilityOptions(options)
    .build();
```

Android - Kotlin

You can use the `TransferUtilityOptions` object to customize the operations of the `TransferUtility`.

TransferThreadPoolSize This parameter will let you specify the number of threads in the thread pool for transfers. By increasing the number of threads, you will be able to increase the number of parts of a mulit-part upload that will be uploaded in parallel. By default, this is set to 2 * (N + 1), where N is the number of available processors on the mobile device. The minimum allowed value is 2.

```
val options = new TransferUtilityOptions().apply {
    transferThreadPoolSize = 8
}

val transferUtility = TransferUtility.builder()
    // Pass-in S3Client, Context, AWSConfiguration/DefaultBucket Name
    .transferUtilityOptions(options)
    .build()
```

TransferServiceCheckTimeInterval The `TransferUtility` monitors each on-going transfer by checking its status periodically. If a stalled transfer is detected, it will be automatically resumed by the `TransferUtility`. The TransferServiceCheckTimeInterval option allows you to set the time interval between the status checks. It is specified in milliseconds and set to 60,000 by default.

```
val options = new TransferUtilityOptions().apply {
    transferServiceCheckTimeInterval = 2 * 60 * 1000 // 2-minutes
}

val transferUtility = TransferUtility.builder()
    // Pass-in S3Client, Context, AWSConfiguration/DefaultBucket Name
    .transferUtilityOptions(options)
    .build()
```

iOS - Swift

You can use the `AWSS3TransferUtilityConfiguration` object to configure the operations of the `TransferUtility`.

isAccelerateModeEnabled The isAccelerateModeEnabled option lets you to upload and download content from a bucket that has Transfer Acceleration enabled on it. See https://docs.aws.amazon.com/AmazonS3/latest/dev/transfer-acceleration.html for information on how to enable transfer acceleration for your bucket.

This option is set to false by default.

```
//Setup credentials
let credentialProvider = AWSCognitoCredentialsProvider(regionType: YOUR-IDENTITY-POOL-
REGION, identityPoolId: "YOUR-IDENTITY-POOL-ID")

//Setup the service configuration
```

```
let configuration = AWSServiceConfiguration(region: .USEast1, credentialsProvider:
 credentialProvider)

//Setup the transfer utility configuration
let tuConf = AWSS3TransferUtilityConfiguration()
tuConf.isAccelerateModeEnabled = true


//Register a transfer utility object
AWSS3TransferUtility.register(
    with: configuration!,
    transferUtilityConfiguration: tuConf,
    forKey: "transfer-utility-with-advanced-options"
)


//Look up the transfer utility object from the registry to use for your transfers.
let transferUtility = AWSS3TransferUtility.s3TransferUtility(forKey: "transfer-utility-
with-advanced-options")
```

- `YOUR-IDENTITY-POOL-REGION` should be in the form of `.USEast1`
- `YOUR-IDENTITY-POOL-ID` should be in the form of `us-east-1:01234567-yyyy-0123-xxxx-012345678901`

retryLimit The retryLimit option allows you to specify the number of times the TransferUtility will retry a transfer when it encounters an error during the transfer. By default, it is set to 0, which means that there will be no retries.

```
tuConf.retryLimit = 5
```

multiPartConcurrencyLimit The multiPartConcurrencyLimit option allows you to specify the number of parts that will be uploaded in parallel for a MultiPart upload request. By default, this is set to 5.

```
tuConf.multiPartConcurrencyLimit = 3
```

More Transfer Examples

Topics

This section provides descriptions and abbreviated examples of the aspects of each type of transfer that are unique. For information about typical code surrounding the following snippets see Track Transfer Progress (p. 201).

Downloading to a File

The following code shows how to download an Amazon S3 Object to a local file.

Android - Java

```
TransferObserver downloadObserver =
    transferUtility.download(
```

```
          "s3Folder/s3Key.txt",
          new File("/path/to/file/localFile.txt"));

downloadObserver.setTransferListener(new TransferListener() {

     @Override
     public void onStateChanged(int id, TransferState state) {
        if (TransferState.COMPLETED == state) {
           // Handle a completed download.
        }
     }

     @Override
     public void onProgressChanged(int id, long bytesCurrent, long bytesTotal) {
           float percentDonef = ((float)bytesCurrent/(float)bytesTotal) * 100;
           int percentDone = (int)percentDonef;

           Log.d("MainActivity", "   ID:" + id + "   bytesCurrent: " + bytesCurrent + "
   bytesTotal: " + bytesTotal + " " + percentDone + "%");
     }

     @Override
     public void onError(int id, Exception ex) {
        // Handle errors
     }

});
```

Android - Kotlin

```
val observer = transferUtility.download(
          "s3Folder/s3Key.txt",
          new File("/path/to/file/localFile.txt"))
observer.transferListener = object : TransferListener() {
    override fun onStateChanged(id: int, state: TransferState) {
        if (state == TransferState.COMPLETED) {
            // Handle a completed download
        }
    }

    override fun onProgressChanged(id: Int, current: Long, total: Long) {
        val done = ((current / total) * 100.0) as Int
        // Do something
    }

    override fun onError(id: Int, ex: Exception) {
        // Do something
    }
}
```

iOS - Swift

```
let fileURL = // The file URL of the download destination.

let transferUtility = AWSS3TransferUtility.default()
transferUtility.download(
        to: fileURL
        bucket: S3BucketName,
        key: S3DownloadKeyName,
        expression: expression,
        completionHandler: completionHandler).continueWith {
            (task) -> AnyObject! in if let error = task.error {
                print("Error: \(error.localizedDescription)")
            }
```

```
            if let _ = task.result {
                // Do something with downloadTask.
            }
            return nil;
        }
```

Uploading Binary Data to a File

Android - Java

Use the following code to upload binary data to a file in Amazon S3.

```
TransferObserver uploadObserver =
        transferUtility.upload(
              "s3Folder/s3Key.bin",
              new File("/path/to/file/localFile.bin"));

uploadObserver.setTransferListener(new TransferListener() {

     @Override
     public void onStateChanged(int id, TransferState state) {
        if (TransferState.COMPLETED == state) {
           // Handle a completed upload.
        }
     }

     @Override
     public void onProgressChanged(int id, long bytesCurrent, long bytesTotal) {
           float percentDonef = ((float)bytesCurrent/(float)bytesTotal) * 100;
           int percentDone = (int)percentDonef;

           Log.d("MainActivity", "   ID:" + id + "   bytesCurrent: " + bytesCurrent + "
   bytesTotal: " + bytesTotal + " " + percentDone + "%");
     }

     @Override
     public void onError(int id, Exception ex) {
        // Handle errors
     }

});
```

Android - Kotlin

Use the following code to upload binary data to a file in Amazon S3.

```
val observer = transferUtility.upload(
          "s3Folder/s3Key.bin",
          new File("/path/to/file/localFile.bin"))
observer.transferListener = object : TransferListener() {
    override fun onStateChanged(id: int, state: TransferState) {
        if (state == TransferState.COMPLETED) {
            // Handle a completed download
        }
    }

    override fun onProgressChanged(id: Int, current: Long, total: Long) {
        val done = ((current / total) * 100.0) as Int
        // Do something
    }
```

```
    override fun onError(id: Int, ex: Exception) {
        // Do something
    }
}
```

iOS - Swift

To upload a binary data to a file, you have to make sure to set the appropriate content type in the uploadData method of the TransferUtility. In the example below, we are uploading a PNG image to S3.

```
let data: Data = Data() // The data to upload

let transferUtility = AWSS3TransferUtility.default()
transferUtility.uploadData(data,
            bucket: S3BucketName,
            key: S3UploadKeyName,
            contentType: "image/png",
            expression: expression,
            completionHandler: completionHandler).continueWith { (task) -> AnyObject!
 in
                if let error = task.error {
                    print("Error: \(error.localizedDescription)")
                }

                if let _ = task.result {
                    // Do something with uploadTask.
                }

                return nil;
            }
```

Downloading Binary Data to a File

The following code shows how to download a binary file.

Android - Java

```
TransferObserver downloadObserver =
    transferUtility.download(
          "s3Folder/s3Key.bin",
          new File("/path/to/file/localFile.bin"));

downloadObserver.setTransferListener(new TransferListener() {

     @Override
     public void onStateChanged(int id, TransferState state) {
        if (TransferState.COMPLETED == state) {
           // Handle a completed download.
        }
     }
     @Override
     public void onProgressChanged(int id, long bytesCurrent, long bytesTotal) {
           float percentDonef = ((float)bytesCurrent/(float)bytesTotal) * 100;
           int percentDone = (int)percentDonef;

           Log.d("MainActivity", "   ID:" + id + "   bytesCurrent: " + bytesCurrent + "
   bytesTotal: " + bytesTotal + " " + percentDone + "%");
     }

     @Override
     public void onError(int id, Exception ex) {
```

```
        // Handle errors
    }

});
```

Android - Kotlin

```
val observer = transferUtility.download(
          "s3Folder/s3Key.bin",
          new File("/path/to/file/localFile.bin"))
observer.transferListener = object : TransferListener() {
    override fun onStateChanged(id: int, state: TransferState) {
        if (state == TransferState.COMPLETED) {
            // Handle a completed download
        }
    }

    override fun onProgressChanged(id: Int, current: Long, total: Long) {
        val done = ((current / total) * 100.0) as Int
        // Do something
    }

    override fun onError(id: Int, ex: Exception) {
        // Do something
    }
}
```

iOS - Swift

```
let fileURL = // The file URL of the download destination
let transferUtility = AWSS3TransferUtility.default()
transferUtility.downloadData(
        fromBucket: S3BucketName,
        key: S3DownloadKeyName,
        expression: expression,
        completionHandler: completionHandler).continueWith {
            (task) -> AnyObject! in if let error = task.error {
                print("Error: \(error.localizedDescription)")
            }

            if let _ = task.result {
                // Do something with downloadTask.
            }

            return nil;
        }
```

Limitations

Android - Java

If you expect your app to perform transfers that take longer than 50 minutes, use AmazonS3Client instead of TransferUtility.

`TransferUtility` generates Amazon S3 pre-signed URLs to use for background data transfer. Using Amazon Cognito Identity, you receive AWS temporary credentials. The credentials are valid for up to 60 minutes. Generated Amazon S3 pre-signed URLs cannot last longer than that time. Because of this limitation, the Amazon S3 Transfer Utility enforces 50 minute transfer timeouts, leaving a 10 minute buffer before AWS temporary credentials are regenerated. After **50 minutes**, you receive a transfer failure.

Android - Kotlin

If you expect your app to perform transfers that take longer than 50 minutes, use AmazonS3Client instead of TransferUtility.

`TransferUtility` generates Amazon S3 pre-signed URLs to use for background data transfer. Using Amazon Cognito Identity, you receive AWS temporary credentials. The credentials are valid for up to 60 minutes. Generated Amazon S3 pre-signed URLs cannot last longer than that time. Because of this limitation, the Amazon S3 Transfer Utility enforces 50 minute transfer timeouts, leaving a 10 minute buffer before AWS temporary credentials are regenerated. After **50 minutes**, you receive a transfer failure.

iOS - Swift

If you expect your app to perform transfers that take longer than 50 minutes, use AWSS3 instead of AWSS3TransferUtility.

`AWSS3TransferUtility` generates Amazon S3 pre-signed URLs to use for background data transfer. Using Amazon Cognito Identity, you receive AWS temporary credentials. The credentials are valid for up to 60 minutes. At the same time, generated S3 pre-signed URLs cannot last longer than that time. Because of this limitation, the AWSS3TransferUtility enforces **50 minutes** transfer timeout, leaving a 10 minute buffer before AWS temporary credentials are regenerated. After 50 minutes, you receive a transfer failure.

Amazon S3 Pre-Signed URLs: For Background Transfer

If you are working with large file transfers, you may want to perform uploads and downloads in the background. To do this, you need to create a background session using `NSURLSession` and then transfer your objects using pre-signed URLs.

The following sections describe pre-signed S3 URLs. To learn more about `NSURLSession`, see Using NSURLSession.

Pre-Signed URLs

By default, all Amazon S3 resources are private. If you want your users to have access to Amazon S3 buckets or objects, you can assign appropriate permissions with an IAM policy.

Alternatively, you can use pre-signed URLs to give your users access to Amazon S3 objects. A pre-signed URL provides access to an object without requiring AWS security credentials or permissions.

When you create a pre-signed URL, you must provide your security credentials, specify a bucket name, an object key, an HTTP method, and an expiration date and time. The pre-signed URL is valid only for the specified duration.

Build a Pre-Signed URL

The following example shows how to build a pre-signed URL for an Amazon S3 download in the background.

iOS - Swift

```
AWSS3PreSignedURLBuilder.default().getPreSignedURL(getPreSignedURLRequest).continueWith
 { (task:AWSTask<NSURL>) -> Any? in
    if let error = task.error as? NSError {
        print("Error: \(error)")
        return nil
    }

    let presignedURL = task.result
    print("Download presignedURL is: \(presignedURL)")
```

```
    let request = URLRequest(url: presignedURL as! URL)
    let downloadTask: URLSessionDownloadTask = URLSession.shared.downloadTask(with:
 request)
    downloadTask.resume()

    return nil
}
```

iOS - Objective-C

```
AWSS3GetPreSignedURLRequest *getPreSignedURLRequest = [AWSS3GetPreSignedURLRequest
 new];
getPreSignedURLRequest.bucket = @"myBucket";
getPreSignedURLRequest.key = @"myImage.jpg";
getPreSignedURLRequest.HTTPMethod = AWSHTTPMethodGET;
getPreSignedURLRequest.expires = [NSDate dateWithTimeIntervalSinceNow:3600];

[[[AWSS3PreSignedURLBuilder defaultS3PreSignedURLBuilder]
 getPreSignedURL:getPreSignedURLRequest]
    continueWithBlock:^id(AWSTask *task) {

    if (task.error) {
        NSLog(@"Error: %@",task.error);
    } else {

        NSURL *presignedURL = task.result;
        NSLog(@"download presignedURL is: \n%@", presignedURL);

        NSURLRequest *request = [NSURLRequest requestWithURL:presignedURL];
        self.downloadTask = [self.session downloadTaskWithRequest:request];
        //downloadTask is an instance of NSURLSessionDownloadTask.
        //session is an instance of NSURLSession.
        [self.downloadTask resume];

    }
    return nil;
}];
```

The preceding example uses `GET` as the HTTP method: `AWSHTTPMethodGET`. For an upload request to Amazon S3, we would need to use a PUT method and also specify a content type.

iOS - Swift

```
getPreSignedURLRequest.httpMethod = .PUT
let fileContentTypeStr = "text/plain"
getPreSignedURLRequest.contentType = fileContentTypeStr
```

iOS - Objective-C

```
getPreSignedURLRequest.HTTPMethod = AWSHTTPMethodPUT;
NSString *fileContentTypeStr = @"text/plain";
getPreSignedURLRequest.contentType = fileContentTypeStr;
```

Here's an example of building a pre-signed URL for a background upload to S3.

iOS - Swift

```
let getPreSignedURLRequest = AWSS3GetPreSignedURLRequest()
```

```
getPreSignedURLRequest.bucket = "myBucket"
getPreSignedURLRequest.key = "myFile.txt"
getPreSignedURLRequest.httpMethod = .PUT
getPreSignedURLRequest.expires = Date(timeIntervalSinceNow: 3600)

//Important: set contentType for a PUT request.
let fileContentTypeStr = "text/plain"
getPreSignedURLRequest.contentType = fileContentTypeStr

AWSS3PreSignedURLBuilder.default().getPreSignedURL(getPreSignedURLRequest).continueWith
 { (task:AWSTask<NSURL>) -> Any? in
    if let error = task.error as? NSError {
        print("Error: \(error)")
        return nil
    }

    let presignedURL = task.result
    print("Download presignedURL is: \(presignedURL)")

    var request = URLRequest(url: presignedURL as! URL)
    request.cachePolicy = .reloadIgnoringLocalCacheData
    request.httpMethod = "PUT"
    request.setValue(fileContentTypeStr, forHTTPHeaderField: "Content-Type")

    let uploadTask: URLSessionTask = URLSession.shared.uploadTask(with: request,
 fromFile: URL(fileURLWithPath: "your/file/path/myFile.txt"))
    uploadTask.resume()

    return nil
}
```

iOS - Objective-C

```
AWSS3GetPreSignedURLRequest *getPreSignedURLRequest = [AWSS3GetPreSignedURLRequest
 new];
getPreSignedURLRequest.bucket = @"myBucket";
getPreSignedURLRequest.key = @"myFile";
getPreSignedURLRequest.HTTPMethod = AWSHTTPMethodPUT;
getPreSignedURLRequest.expires = [NSDate dateWithTimeIntervalSinceNow:3600];

//Important: set contentType for a PUT request.
NSString *fileContentTypeStr = @"text/plain";
getPreSignedURLRequest.contentType = fileContentTypeStr;

[[[AWSS3PreSignedURLBuilder defaultS3PreSignedURLBuilder]
 getPreSignedURL:getPreSignedURLRequest]
                                                 continueWithBlock:^id(AWSTask *task) {
    if (task.error) {
        NSLog(@"Error: %@",task.error);
    } else {
        NSURL *presignedURL = task.result;
        NSLog(@"upload presignedURL is: \n%@", presignedURL);

        NSMutableURLRequest *request = [NSMutableURLRequest
 requestWithURL:presignedURL];
        request.cachePolicy = NSURLRequestReloadIgnoringLocalCacheData;
        [request setHTTPMethod:@"PUT"];
        [request setValue:fileContentTypeStr forHTTPHeaderField:@"Content-Type"];

        self.uploadTask = [self.session uploadTaskWithRequest:request
 fromFile:self.uploadFileURL];
        //uploadTask is an instance of NSURLSessionDownloadTask.
        //session is an instance of NSURLSession.
        [self.uploadTask resume];
    }
```

```
    return nil;
}];
```

Additional Resources

- Amazon Simple Storage Service Getting Started Guide
- Amazon Simple Storage Service API Reference
- Amazon Simple Storage Service Developer Guide

Amazon S3 Server-Side Encryption Support in iOS

The AWS Mobile SDK for iOS supports server-side encryption of Amazon S3 data. To learn more about server-side encryption, see PUT Object.

The following properties are available to configure the encryption:

- SSECustomerAlgorithm
- SSECustomerKey
- SSECustomerKeyMD5
- AWSS3ServerSideEncryption

To use these properties, import the `AWSSS3Model` with the following statement.

iOS - Swift

```
import AWSS3
```

iOS - Objective-C

```
#import <AWSS3/AWSS3.h>
```

`SSECustomerAlgorithm` is a property of `AWSS3ReplicateObjectOutput`. If server-side encryption with a customer-provided encryption key was requested, the response will include this header, which confirms the encryption algorithm that was used. Currently, the only valid option is AES256. You can access `SSECustomerAlgorithm` as follows.

iOS - Swift

```
let replicateObjectOutput = AWSS3ReplicateObjectOutput()
replicateObjectOutput?.sseCustomerAlgorithm = "mySseCustomerAlgorithm
```

iOS - Objective-C

```
AWSS3ReplicateObjectOutput *replicateObjectOutput = [AWSS3ReplicateObjectOutput new];
replicateObjectOutput.SSECustomerAlgorithm = @"mySseCustomerAlgorithm";
```

`SSECustomerKey`, a property of `AWSS3UploadPartRequest`, specifies the customer-provided encryption key for Amazon S3 to use to encrypting data. This value is used to store the object, and is then discarded; Amazon doesn't store the encryption key. The key must be appropriate for use with the algorithm specified in the `x-amz-server-side-encryption-customer-algorithm` header. This

must be the same encryption key specified in the request to initiate a multipart upload. You can access SSECustomerKey as follows.

iOS - Swift

```
let uploadPartRequest = AWSS3UploadPartRequest()
uploadPartRequest?.sseCustomerKey = "customerProvidedEncryptionKey"
```

iOS - Objective-C

```
AWSS3UploadPartRequest *uploadPartRequest = [AWSS3UploadPartRequest new];
uploadPartRequest.SSECustomerKey = @"customerProvidedEncryptionKey";
```

`SSECustomerKeyMD5` is a property of `AWSS3PutObjectOutput`. If server-side encryption with a customer-provided encryption key is requested, the response will include this header. The response provides round trip message integrity verification of the customer-provided encryption key. You can access `SSECustomerKeyMD5` as follows.

iOS - Swift

```
let objectOutput = AWSS3PutObjectOutput()
// Access objectOutput?.sseCustomerKeyMD5 ...
```

iOS - Objective-C

```
AWSS3PutObjectOutput *objectOutput = [AWSS3PutObjectOutput new];
//Access objectOutput.SSECustomerKeyMD5 ...
```

`AWSS3ServerSideEncryption` represents the encryption algorithm for storing an object in Amazon S3. You can access it as follows.

iOS - Swift

```
let objectOutput = AWSS3PutObjectOutput()
// Access objectOutput?.sseCustomerKeyMD5 ...
```

iOS - Objective-C

```
AWSS3ReplicateObjectOutput *replicateObjectOutput = [AWSS3ReplicateObjectOutput new];
// Access replicateObjectOutput.serverSideEncryption ...
```

Additional Resources

- Amazon Simple Storage Service Getting Started Guide
- Amazon Simple Storage Service API Reference
- Amazon Simple Storage Service Developer Guide

iOS: Amazon S3 TransferManager for iOS

Just Getting Started?	Use streamlined steps (p. 66) to install the SDK and integrate Amazon S3.

Or, use the contents of this page if your app will integrate existing AWS services.

Topics

Amazon Simple Storage Service (S3)

Amazon Simple Storage Service (S3) provides secure, durable, highly-scalable object storage in the cloud. Using the AWS Mobile SDK for iOS, you can directly access Amazon S3 from your mobile app. For information about Amazon S3 regional availability, see AWS Service Region Availability.

TransferManager Features

Amazon S3 TransferManager class makes it easy to upload files to and download files from Amazon S3 while optimizing for performance and reliability. It hides the complexity of transferring files behind a simple API.

Whenever possible, uploads are broken into multiple pieces, so that several pieces are sent in parallel to provide better throughput. This approach enables more robust transfers, since an I/O error in an individual piece result in the SDK retransmitting only the faulty piece, not the entire transfer. `TransferManager` provides simple APIs to pause, resume, and cancel file transfers.

The following sections provide a step-by-step guide for getting started with Amazon S3 using the `TransferManager`.

You can also try out the Amazon S3 sample available in the AWSLabs GitHub repository.
<admonition>
<title>Should I Use `TransferManager` or `TransferUtility`?</title>

To choose which API best suits your needs, see manager-or-utility.
</admonition>

Setup

To set your project up to use the `TransferManager` class, take the steps below.

1. Setup the SDK, Credentials and Services

Follow the steps in How to Integrate Your Existing Bucket (p. 182) to install the AWS Mobile SDK for iOS and configure AWS credentials and permissions.

2. Import the SDK Amazon S3 APIs

Add the following import statements to your Xcode project.

iOS - Swift

```
import AWSS3
```

iOS - Objective-C

```
#import <AWSS3/AWSS3.h>
```

3. Create the `TransferManager` Client

Add the following code to create an *AWSS3TransferManager* client.

iOS - Swift

```
let transferManager = AWSS3TransferManager.default()
```

iOS - Objective-C

```
AWSS3TransferManager *transferManager = [AWSS3TransferManager
 defaultS3TransferManager];
```

The *AWSS3TransferManager* class is an entry point to this SDK's high-level Amazon S3 APIs.

Transfer an Object

In this section:

Downloading a file from and uploading a file to a bucket, use the same coding pattern. An important difference is that *download:* does not succeed until the download is complete, blocking any flow that depends on that success. Upload returns immediately and can therefore be safely called on the main thread.

The steps to call `TransferManager` for a transfer are as follows.

1. Create an `AWSS3TransferManagerDownloadRequest`

The following code illustrates the three actions needed to create a download request:

- Create a destination/source location for the file. In this example, this is called `downloadingFileURL` / `uploadingFileURL`.
- Construct a request object using `AWSS3TransferManagerDownloadRequest`.
- Set three properties of the request object: the bucket name; the key (the name of the object in the bucket); and the download destination / upload source `downloadingFileURL` / `uploadingFileURL`.

Download

iOS - Swift

```
let downloadingFileURL = URL(fileURLWithPath:
 NSTemporaryDirectory()).appendingPathComponent("myImage.jpg")

let downloadRequest = AWSS3TransferManagerDownloadRequest()

downloadRequest.bucket = "myBucket"
downloadRequest.key = "myImage.jpg"
downloadRequest.downloadingFileURL = downloadingFileURL
```

iOS - Objective-C

```
NSString *downloadingFilePath = [NSTemporaryDirectory()
 stringByAppendingPathComponent:@"myImage.jpg"];
NSURL *downloadingFileURL = [NSURL fileURLWithPath:downloadingFilePath];

AWSS3TransferManagerDownloadRequest *downloadRequest =
 [AWSS3TransferManagerDownloadRequest new];
```

```
downloadRequest.bucket = @"myBucket";
downloadRequest.key = @"myImage.jpg";
downloadRequest.downloadingFileURL = downloadingFileURL;
```

Upload

iOS - Swift

```
let uploadingFileURL = URL(fileURLWithPath: "your/file/path/myTestFile.txt")

let uploadRequest = AWSS3TransferManagerUploadRequest()

uploadRequest.bucket = "myBucket"
uploadRequest.key = "myTestFile.txt"
uploadRequest.body = uploadingFileURL
```

iOS - Objective-C

```
NSURL *uploadingFileURL = [NSURL fileURLWithPath: @"your/file/path/myTestFile.txt"];

AWSS3TransferManagerUploadRequest *uploadRequest = [AWSS3TransferManagerUploadRequest
 new];

uploadRequest.bucket = @"myBucket";
uploadRequest.key = @"myTestFile.txt";
uploadRequest.body = uploadingFileURL;
```

2. Pass the Request to the *download:* Method

Use the following code to pass the request to the *download: / upload:" method of the ``TransferManager`* client. The methods are asynchronous and returns an *AWSTask* object. Use a *continueWith* block to handle the method result. For more information about *AWSTask*, see Working with Asynchronous Tasks.

Download

iOS - Swift

```
transferManager.download(downloadRequest).continueWith(executor:
 AWSExecutor.mainThread(), block: { (task:AWSTask<AnyObject>) -> Any? in

    if let error = task.error as? NSError {
        if error.domain == AWSS3TransferManagerErrorDomain, let code =
 AWSS3TransferManagerErrorType(rawValue: error.code) {
            switch code {
            case .cancelled, .paused:
                break
            default:
                print("Error downloading: \(downloadRequest.key) Error: \(error)")
            }
        } else {
            print("Error downloading: \(downloadRequest.key) Error: \(error)")
        }
        return nil
    }
    print("Download complete for: \(downloadRequest.key)")
    let downloadOutput = task.result
    return nil
```

```
})
```

iOS - Objective-C

```
[[transferManager download:downloadRequest ] continueWithExecutor:[AWSExecutor
 mainThreadExecutor]
    withBlock:^id(AWSTask *task) {
    if (task.error){
        if ([task.error.domain isEqualToString:AWSS3TransferManagerErrorDomain]) {
            switch (task.error.code) {
                case AWSS3TransferManagerErrorCancelled:
                case AWSS3TransferManagerErrorPaused:
                break;

                default:
                    NSLog(@"Error: %@", task.error);
                    break;
            }

        } else {
            NSLog(@"Error: %@", task.error);
        }
    }

    if (task.result) {
        AWSS3TransferManagerDownloadOutput *downloadOutput = task.result;
    }
    return nil;
}];
```

Upload

iOS - Swift

```
transferManager.upload(uploadRequest).continueWith(executor: AWSExecutor.mainThread(),
 block: { (task:AWSTask<AnyObject>) -> Any? in

    if let error = task.error as? NSError {
        if error.domain == AWSS3TransferManagerErrorDomain, let code =
 AWSS3TransferManagerErrorType(rawValue: error.code) {
            switch code {
            case .cancelled, .paused:
                break
            default:
                print("Error uploading: \(uploadRequest.key) Error: \(error)")
            }
        } else {
            print("Error uploading: \(uploadRequest.key) Error: \(error)")
        }
        return nil
    }

    let uploadOutput = task.result
    print("Upload complete for: \(uploadRequest.key)")
    return nil
})
```

iOS - Objective-C

```
[[transferManager upload:uploadRequest] continueWithExecutor:[AWSExecutor
 mainThreadExecutor]
            withBlock:^id(AWSTask *task) {
```

```
if (task.error) {
    if ([task.error.domain isEqualToString:AWSS3TransferManagerErrorDomain]) {
        switch (task.error.code) {
            case AWSS3TransferManagerErrorCancelled:
            case AWSS3TransferManagerErrorPaused:
                break;

            default:
                NSLog(@"Error: %@", task.error);
                break;
        }
    } else {
        // Unknown error.
        NSLog(@"Error: %@", task.error);
    }
}

if (task.result) {
    AWSS3TransferManagerUploadOutput *uploadOutput = task.result;
    // The file uploaded successfully.
}
return nil;
```

}];

3. Displaying a Downloaded Image in an UIImageView

The use of *download:* in this example is executed on the main thread. The following code illustrates displaying such an image in a *UIImageView* configured in your project .

Note that it can only succeed after download of the file it displays has completed.

iOS - Swift

```
self.imageView.image = UIImage(contentsOfFile: downloadingFileURL.path)
```

iOS - Objective-C

```
self.imageView.image = [UIImage imageWithContentsOfFile:downloadingFilePath];
```

Pause, Resume, and Cancel Object Transfers

In this section:

Topics

The `TransferManager` supports pause, resume, and cancel operations for both uploads and downloads. The *pause*, *cancel*, *resumeAll*, *cancelAll*, *pauseAll*, *upload:*, and *download:* operations all return instances of *AWSTask*. Use these methods with a *continueWith block:* to handle the returns of these operations.

Use continueWith Block to Handle Results

The following code illustrates using *continueWith block:* when calling the *pause* method.

iOS - Swift

```
uploadRequest.pause().continueWith(block: { (task:AWSTask<AnyObject>) -> Any? in
    if let error = task.error as? NSError {
        print("Error: \(error)")
        return nil
    }

    // Upload has been paused.
    return nil
})
```

iOS - Objective-C

```
[[self.uploadRequest pause] continueWithBlock:^id(AWSTask *task) {
    if (task.error) {
        NSLog(@"Error: %@",task.error);
    } else {

    }

    // Upload has been paused.
    return nil;
}];
```

For brevity, the following examples omit the *continueWithBlock*.

Pause a Transfer

To pause an object transfer, call *pause* on the request object.

iOS - Swift

```
uploadRequest.pause()
downloadRequest.pause()
```

iOS - Objective-C

```
[uploadRequest pause];
[downloadRequest pause];
```

Resume a Transfer

To resume a transfer, call *upload* or *download* and pass in the paused request object.

iOS - Swift

```
transferManager.upload(uploadRequest)
transferManager.download(downloadRequest)
```

iOS - Objective-C

```
[transferManager upload:uploadRequest];
```

```
[transferManager download:downloadRequest];
```

Cancel a Transfer

To cancel a transfer, call *cancel* on the upload or download request.

iOS - Swift

```
uploadRequest.cancel()
downloadRequest.cancel()
```

iOS - Objective-C

```
[uploadRequest cancel];
[downloadRequest cancel];
```

Pause All Transfers

To pause all of the current upload and download requests, call *pauseAll* on the `TransferManager`.

iOS - Swift

```
transferManager.pauseAll()
```

iOS - Objective-C

```
[transferManager pauseAll];
```

Resume All Transfers

To resume all of the current upload and download requests, call *resumeAll* on the `TransferManager` passing an *AWSS3``TransferManager``ResumeAllBlock*, which is a closure that takes *AWSRequest* as a parameter, and can be used to reset the progress blocks for the requests.

iOS - Swift

```
transferManager.resumeAll({ (request:AWSRequest?) in
   // All paused requests have resumed.
})
```

iOS - Objective-C

```
[transferManager resumeAll:^(AWSRequest *request) {
    // All paused requests have resumed.
}];
```

Cancel All Transfers

To cancel all upload and download requests, call *cancelAll* on the TransferManager.

iOS - Swift

```
transferManager.cancelAll()
```

iOS - Objective-C

```
[transferManager cancelAll];
```

Track Progress

Using the *uploadProgress* and *downloadProgress* blocks, you can track the progress of object transfers. These blocks work in conjunction with the Grand Central Dispatch *dispatch_async* function, as shown in the following examples.

Upload Progress

Track the progress of an upload.

iOS - Swift

```
uploadRequest.uploadProgress = {(bytesSent: Int64, totalBytesSent: Int64,
 totalBytesExpectedToSend: Int64) -> Void in
    DispatchQueue.main.async(execute: {() -> Void in
        //Update progress
    })
}
```

iOS - Objective-C

```
uploadRequest.uploadProgress =  ^(int64_t bytesSent, int64_t totalBytesSent, int64_t
 totalBytesExpectedToSend){
    dispatch_async(dispatch_get_main_queue(), ^{
     //Update progress
});
```

Download Progress

Track the progress of a download.

iOS - Swift

```
downloadRequest.downloadProgress = {(bytesSent: Int64, totalBytesSent: Int64,
 totalBytesExpectedToSend: Int64) -> Void in
    DispatchQueue.main.async(execute: {() -> Void in
        //Update progress
    })
}
```

iOS - Objective-C

```
downloadRequest.downloadProgress = ^(int64_t bytesWritten, int64_t totalBytesWritten,
 int64_t totalBytesExpectedToWrite){
dispatch_async(dispatch_get_main_queue(), ^{
        //Update progress
});
```

Multipart Upload

Amazon S3 provides a multipart upload feature to upload a single object as a set of parts. Each part is a contiguous portion of the object's data. The object parts are uploaded independently and in any order. If

transmission of any part fails, you can retransmit that part without affecting other parts. After all parts of the object are uploaded, Amazon S3 assembles these parts and creates the object.

In the AWS Mobile SDK for iOS, the `TransferManager` handles multipart upload for you. The minimum part size for a multipart upload is 5MB.

Additional Resources

- Amazon Simple Storage Service Getting Started Guide
- Amazon Simple Storage Service API Reference
- Amazon Simple Storage Service Developer Guide

How To: NoSQL Database with Amazon DynamoDB

Just Getting Started?	Use streamlined steps (p. 54) to install the SDK and integrate Amazon DynamoDB.

This section provides information on the steps for achieving specific tasks for integrating your DynamoDB into your Android and iOS apps.

Topics

Integrate Your Existing NoSQL Table

Just Getting Started?	Use streamlined steps (p. 54) to install the SDK and integrate features.

The Get Started (p. 54) section of this guide allows you to create new resources and complete the steps described on this page in minutes. If you want to import existing resources or create them from scratch, the contents of this page will walk you through the steps you need.

The following steps and examples are based on a simple bookstore app. The app tracks the books that are available in the bookstore using an Amazon DynamoDB table.

Set up Your Backend

To manually configure an Amazon DynamoDB table that you can integrate into your mobile app, use the following steps.

Topics

Create an New Table and Index

- If you already have an Amazon DynamoDB table and know its region, you can skip to Set Up an Identity Pool (p. 233).

To create the Books table:

1. Sign in to the Amazon DynamoDB Console.
2. Choose **Create Table**.
3. Type `Books` as the name of the table.
4. Enter `ISBN` in the **Partition key** field of the **Primary key** with **String** as their type.
5. Check the **Add sort key** box , then type `Category` in the provided field and select **String** as the type.
6. Clear the **Use default settings** checkbox and choose **+ Add Index**.
7. In the **Add Index** dialog type **`Author`** with **String** as the type.
8. Check the **Add sort key** checkbox and enter **`Title`** as the sort key value, with **String** as its type.
9. Leave the other values at their defaults. Choose **Add index** to add the **`Author-Title-index`** index.
10. Set the **Minimum provisioned capacity** for read to 10, and for write to 5.
11. Choose **Create**.Amazon DynamoDB will create your database.
12. Refresh the console and choose your Books table from the list of tables.
13. Open the **Overview** tab and copy or note the Amazon Resource Name (ARN). You need this for the next procedure.

Set Up an Identity Pool

To give your users permissions to access your table you'll need an identity pool from Amazon Cognito. That pool has two default IAM roles, one for guest (unauthenticated), and one for signed-in (authenticated) users. The policies you design and attach to the IAM roles determine what each type of user can and cannot do.

Import an existing pool or create a new pool (p. 147) for your app.

Set Permissions

Attach the following IAM policy to the unauthenticated role for your identity pool. It allows the user to perform the actions on two resources (a table and an index) identified by the ARN of your Amazon DynamoDB table.

```
{
    "Statement": [{
        "Effect": "Allow",
        "Action": [
            "dynamodb:DeleteItem",
            "dynamodb:GetItem",
            "dynamodb:PutItem",
            "dynamodb:Scan",
            "dynamodb:Query",
            "dynamodb:UpdateItem",
            "dynamodb:BatchWriteItem"
        ],
        "Resource": [
            "arn:aws:dynamodb:us-west-2:123456789012:table/Books",
            "arn:aws:dynamodb:us-west-2:123456789012:table/Books/index/*"
        ]
    }]
}
```

Apply Permissions

Apply this policy to the unauthenticated role assigned to your Amazon Cognito identity pool, replacing the **Resource** values with the correct ARN for the Amazon DynamoDB table:

1. Sign in to the IAM console.
2. Choose **Roles** and then choose the "Unauth" role that Amazon Cognito created for you.
3. Choose **Attach Role Policy**.
4. Choose **Custom Policy** and then Choose **Select**.
5. Type a name for your policy and paste in the policy document shown above, replacing the *Resource* values with the ARNs for your table and index. (You can retrieve the table ARN from the **Details** tab of the database; then append `/index/*` to obtain the value for the index ARN.
6. Choose **Apply Policy**.

Connect to Your Backend

Topics

Create Your AWS Configuration File

Your app is connected to your AWS resources using an `awsconfiguration.json` file which contains the endpoints for the services you use.

1. Create a file with name `awsconfiguration.json` with the following contents:

```
{
  "Version": "1.0",
  "CredentialsProvider": {
    "CognitoIdentity": {
      "Default": {
        "PoolId": "COGNITO-IDENTITY-POOL-ID",
        "Region": "COGNITO-IDENTITY-POOL-REGION"
      }
    }
  },
  "IdentityManager": {
    "Default": {}
  },
  "DynamoDBObjectMapper": {
    "Default": {
      "Region": "DYNAMODB-REGION"
    }
  }
}
```

2. Make the following changes to the configuration file.
   - Replace the `DYNAMODB-REGION` with the region the table was created in.

Need to find your table's region?	Go to Amazon DynamoDB Console. and choose the **Overview** tab for your table. The **Amazon**

	Resource Name (ARN) item shows the table's ID, which contains its region. For example, if your pool ID is `arn:aws:dynamodb:us-east-1:012345678901:table/nosqltest-mobilehub-012345678-Books`, then your the table's region value would be `us-east-1`. The configuration file value you want is in the form of: `"Region": "REGION-OF-YOU-DYNAMODB-ARN"`. For this example: `"Region": "us-east-1"`

- Replace the `COGNITO-IDENTITY-POOL-ID` with the identity pool ID.
- Replace the `COGNITO-IDENTITY-POOL-REGION` with the region the identity pool was created in.

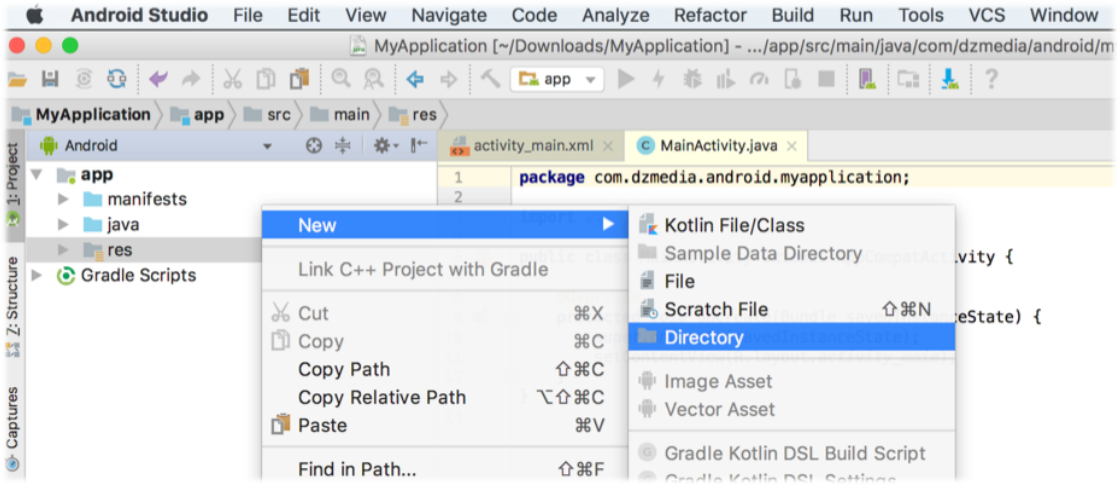
Need to find your pool's ID and region?	Go to Amazon Cognito Console and choose **Manage Federated Identities**, then choose your pool and choose **Edit identity pool**. Copy the value of **Identity pool ID**.
	
	Insert this region value into the following form to create the value you need for this integration.
	
	`"Region": "REGION-PREFIX-OF-YOUR-POOL-ID".`
	
	For example, if your pool ID is `us-east-1:01234567-yyyy-0123-xxxx-012345678901`, then your integration region value would be:
	
	`"Region": "us-east-1"`

Add the AWS Config File

To make the connection between your app and your backend services, add the configuration file.

Android - Java

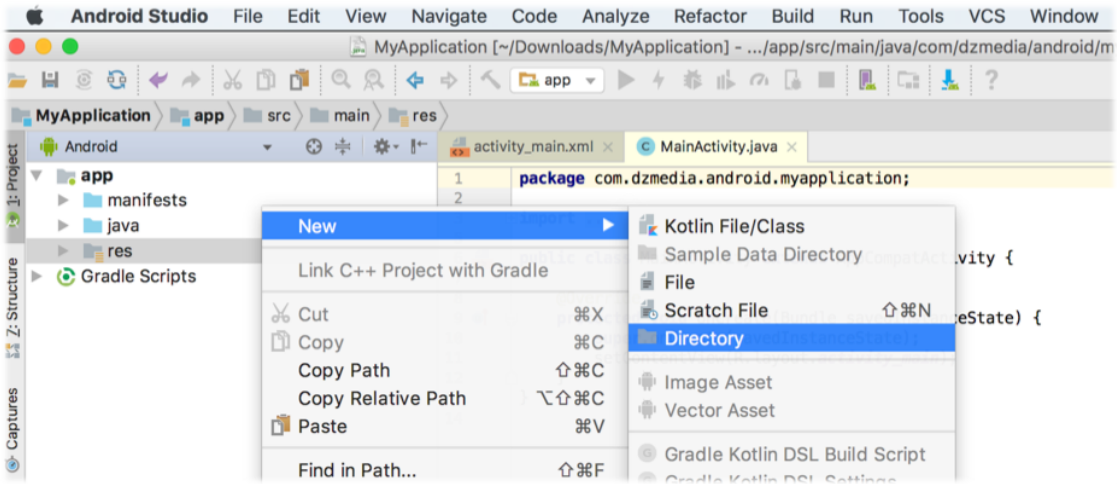
In the Android Studio Project Navigator, right-click your app's `res` folder, and then choose **New > Directory**. Type `raw` as the directory name and then choose **OK**.



Drag the `awsconfiguration.json` you created into the `res/raw` folder. Android gives a resource ID to any arbitrary file placed in this folder, making it easy to reference in the app.

Android - Kotlin

In the Android Studio Project Navigator, right-click your app's `res` folder, and then choose **New > Directory**. Type `raw` as the directory name and then choose **OK**.



Drag the `awsconfiguration.json` you created into the `res/raw` folder. Android gives a resource ID to any arbitrary file placed in this folder, making it easy to reference in the app.

iOS - Swift

Drag the `awsconfiguration.json` into the folder containing your `Info.plist` file in your Xcode project. Choose **Copy items** and **Create groups** in the options dialog.

Add the SDK to your App

Use the following steps to add AWS Mobile NoSQL Database to your app.

Android - Java

1. Set up AWS Mobile SDK components with the following Set Up Your Backend (p. 2) steps.
   a. `app/build.gradle` must contain:

```
dependencies{

    // Amazon Cognito dependencies for user access to AWS resources
    implementation ('com.amazonaws:aws-android-sdk-mobile-client:2.6.+@aar')
 { transitive = true }

    // AmazonDynamoDB dependencies for NoSQL Database
    implementation 'com.amazonaws:aws-android-sdk-ddb-mapper:2.6.+'

    // other dependencies . . .
}
```

b. Add the following permissions to `AndroidManifest.xml`.

```
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
```

2. Create an `AWSDynamoDBMapper` client in the call back of your call to instantiate `AWSMobileClient`. This will ensure that the AWS credentials needed to connect to Amazon DynamoDB are available, and is typically in `onCreate` function of of your start up activity.

```
import com.amazonaws.mobile.client.AWSMobileClient;
import com.amazonaws.mobile.client.AWSStartupHandler;
import com.amazonaws.mobile.client.AWSStartupResult;

import com.amazonaws.mobileconnectors.dynamodbv2.dynamodbmapper.DynamoDBMapper;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClient;

public class MainActivity extends AppCompatActivity {

    // Declare a DynamoDBMapper object
    DynamoDBMapper dynamoDBMapper;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        // AWSMobileClient enables AWS user credentials to access your table
        AWSMobileClient.getInstance().initialize(this, new AWSStartupHandler() {

            @Override
            public void onComplete(AWSStartupResult awsStartupResult) {

                    // Add code to instantiate a AmazonDynamoDBClient
                    AmazonDynamoDBClient dynamoDBClient = new
 AmazonDynamoDBClient(AWSMobileClient.getInstance().getCredentialsProvider());
                    this.dynamoDBMapper = DynamoDBMapper.builder()
                        .dynamoDBClient(dynamoDBClient)
                        .awsConfiguration(
                        AWSMobileClient.getInstance().getConfiguration())
                        .build();

            }
        }).execute();

        // Other functions in onCreate . . .
    }
}
```

Important	**Use Asynchronous Calls to DynamoDB**
	
	Since calls to DynamoDB are synchronous, they don't belong on your UI thread. Use an asynchronous method like the `Runnable` wrapper to call `DynamoDBObjectMapper` in a separate thread.
	
	`Runnable runnable = new Runnable() {`
	`  public void run() {`
	`  //DynamoDB calls go here`
	`  }`
	`};`
	`Thread mythread = new Thread(runnable);`
	`mythread.start();`

Android - Kotlin

1. Set up AWS Mobile SDK components with the following Set Up Your Backend (p. 2) steps.

   a. `app/build.gradle` must contain:

```
dependencies{

    // Amazon Cognito dependencies for user access to AWS resources
    implementation ('com.amazonaws:aws-android-sdk-mobile-client:2.6.+@aar')
 { transitive = true }

    // AmazonDynamoDB dependencies for NoSQL Database
    implementation 'com.amazonaws:aws-android-sdk-ddb-mapper:2.6.+'

    // other dependencies . . .
}
```

   b. Add the following permissions to `AndroidManifest.xml`.

```
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
```

2. Create an `AWSDynamoDBMapper` client in the call back of your call to instantiate `AWSMobileClient`. This will ensure that the AWS credentials needed to connect to Amazon DynamoDB are available, and is typically in `onCreate` function of of your start up activity.

```
import com.amazonaws.mobile.client.AWSMobileClient;
import com.amazonaws.mobile.client.AWSStartupHandler;
import com.amazonaws.mobile.client.AWSStartupResult;

import com.amazonaws.mobileconnectors.dynamodbv2.dynamodbmapper.DynamoDBMapper;
import com.amazonaws.services.dynamodbv2.AmazonDynamoDBClient;

class MainActivity : AppCompatActivity() {
    var ddbMapper: DynamoDBMapper? = null

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        AWSMobileClient.getInstance().initialize(this, object : AWSStartupHandler() {
            override fun onComplete(awsStartupResult: AWSStartupResult) {
```

```
                val ddbClient =
 AmazonDynamoDBClient(AWSMobileClient.getInstance().credentialsProvider)
                ddbMapper = DynamoDBMapper.builder()
                    .dynamoDBClient(ddbClient)
                    .awsConfiguration(AWSMobileClient.getInstance().configuration)
                    .build()
            }
        }).execute()

        // other setup within onCreate() ...
    }
}
```

Important	**Use Asynchronous Calls to DynamoDB**
	
	Since calls to DynamoDB are synchronous, they don't belong on your UI thread. Use an asynchronous method like the `thread` wrapper to call `DynamoDBObjectMapper` in a separate thread.
	
	`thread(start = true ) {`
	`    // DynamoDB calls go here`
	`}`

iOS - Swift

1. Set up AWS Mobile SDK components with the following Set Up Your Backend (p. 2) steps.
   a. Add the `AWSDynamoDB` pod to your `Podfile` to install the AWS Mobile SDK.

```
platform :ios, '9.0'

target :'YOUR-APP-NAME' do
  use_frameworks!

    # Enable AWS user credentials
    pod 'AWSMobileClient', '~> 2.6.13'

    # Connect to NoSQL database tables
    pod 'AWSDynamoDB', '~> 2.6.13'

    # other pods . . .
end
```

   Run `pod install --repo-update` before you continue.

   If you encounter an error message that begins "`[!] Failed to connect to GitHub to update the CocoaPods/Specs . . .`", and your internet connectivity is working, you may need to update openssl and Ruby.

   b. Classes that call DynamoDB APIs must use the following import statements:

```
import AWSCore
import AWSDynamoDB
```

Add Data Models to Your App

To connect your app to your table create a data model object in the following form. In this example, the model is based on the `Books` table you created in a previous step. The partition key (hash key) is called `ISBN` and the sort key (rangekey) is called `Category`.

Android - Java

In the Android Studio project explorer right-click the folder containing your main activity, and choose **New > Java Class**. Type the **Name** you will use to refer to your data model. In this example the name would be `BooksDO`. Add code in the following form.

```
package com.amazonaws.models.nosql;

import com.amazonaws.mobileconnectors.dynamodbv2.dynamodbmapper.DynamoDBAttribute;
import com.amazonaws.mobileconnectors.dynamodbv2.dynamodbmapper.DynamoDBHashKey;
import com.amazonaws.mobileconnectors.dynamodbv2.dynamodbmapper.DynamoDBIndexHashKey;
import com.amazonaws.mobileconnectors.dynamodbv2.dynamodbmapper.DynamoDBIndexRangeKey;
import com.amazonaws.mobileconnectors.dynamodbv2.dynamodbmapper.DynamoDBRangeKey;
import com.amazonaws.mobileconnectors.dynamodbv2.dynamodbmapper.DynamoDBTable;

import java.util.List;
import java.util.Map;
import java.util.Set;

@DynamoDBTable(tableName = "Books")

public class BooksDO {
    private String _isbn;
    private String _category;
    private String _title;
    private String _author;

    @DynamoDBHashKey(attributeName = "ISBN")
    @DynamoDBAttribute(attributeName = "ISBN")
    public String getIsbn() {
        return _isbn;
    }

    public void setIsbn(final String _isbn) {
        this._isbn = _isbn;
    }

    @DynamoDBRangeKey (attributeName = "Category")
    @DynamoDBAttribute(attributeName = "Category")
    public String getCategory() {
        return _category;
    }

    public void setCategory(final String _category) {
        this._category= _category;
    }

    @DynamoDBIndexHashKey(attributeName = "Author", globalSecondaryIndexName =
 "Author")
    public String getAuthor() {
        return _author;
    }

    public void setAuthor(final String _author) {
        this._author = _author;
    }

    @DynamoDBIndexRangeKey(attributeName = "Title", globalSecondaryIndexName = "Title")
```

```
    public String getTitle() {
        return _title;
    }

    public void setTitle(final String _title) {
        this._title = _title;
    }

}
```

Android - Kotlin

In the Android Studio project explorer right-click the folder containing your main activity, and choose **New > Java Class**. Type the **Name** you will use to refer to your data model. In this example the name would be `BooksDO`. Add code in the following form. You can also use a data model in the Java form in a Kotlin project.

```
package com.amazonaws.models.nosql;

import com.amazonaws.mobileconnectors.dynamodbv2.dynamodbmapper.DynamoDBAttribute;
import com.amazonaws.mobileconnectors.dynamodbv2.dynamodbmapper.DynamoDBHashKey;
import com.amazonaws.mobileconnectors.dynamodbv2.dynamodbmapper.DynamoDBIndexHashKey;
import com.amazonaws.mobileconnectors.dynamodbv2.dynamodbmapper.DynamoDBIndexRangeKey;
import com.amazonaws.mobileconnectors.dynamodbv2.dynamodbmapper.DynamoDBRangeKey;
import com.amazonaws.mobileconnectors.dynamodbv2.dynamodbmapper.DynamoDBTable;

import java.util.List;
import java.util.Map;
import java.util.Set;

@DynamoDBTable(tableName = "Books")
class BooksDO {
    @DynamoDBHashKey(attributeName = "ISBN")
    @DynamoDBAttribute(attributeName = "ISBN")
    var isbn: String? = null

    @DynamoDBRangeKey (attributeName = "Category")
    @DynamoDBAttribute(attributeName = "Category")
    var category: String? = null

    @DynamoDBIndexHashKey(attributeName = "Author", globalSecondaryIndexName =
 "Author")
    var author: String? = null

    @DynamoDBIndexRangeKey(attributeName = "Title", globalSecondaryIndexName = "Title")
    var title: String? = null
}
```

iOS - Swift

In the Xcode project explorer, right-click the folder containing your app delegate, and choose **New File > Swift File > Next**. Type the name you will use to refer to your data model as the filenam. In this example the name would be `Books`. Add code in the following form.

```
 import Foundation
 import UIKit
 import AWSDynamoDB

 class Books: AWSDynamoDBObjectModel, AWSDynamoDBModeling {

      @objc var _isbn: String?
      @objc var _category: String?
```

```
    @objc var _author: String?
    @objc var _title: String?

    class func dynamoDBTableName() -> String {
        return "Books"
    }

    class func hashKeyAttribute() -> String {
        return "_isbn"
    }

    class func rangeKeyAttribute() -> String {
        return "_category"
    }

    override class func jsonKeyPathsByPropertyKey() -> [AnyHashable: Any] {
        return [
            "_isbn" : "ISBN",
            "_category" : "Category",
            "_author" : "Author",
            "_title" : "Title",
        ]
    }
}
```

Perform CRUD Operations

The fragments below consume the `BooksDO` data model class created in a previous step.

Topics

Create (Save) an Item

Use the following code to create an item in your NoSQL Database table.

Android - Java

```
public void createBooks() {
    final com.amazonaws.models.nosql.BooksDO booksItem = new
 com.amazonaws.models.nosql.BooksDO();

    booksItem.setIsbn("ISBN1");
    booksItem.setAuthor("Frederick Douglas");
    booksItem.setTitle("Escape from Slavery");
    booksItem.setCategory("History");

    new Thread(new Runnable() {
        @Override
        public void run() {
            dynamoDBMapper.save(booksItem);
            // Item saved
        }
    }).start();
}
```

Android - Kotlin

```
fun createBooks() {
    val booksItem = BooksDO().apply {
        isbn = "ISBN1"
        author = "Frederick Douglas"
        title = "Escape from Slavery"
        category = "History"
    }

    thread(start = true) {
        ddbMapper.save(booksItem)
    }
}
```

iOS - Swift

```
func createBooks() {
    let dynamoDbObjectMapper = AWSDynamoDBObjectMapper.default()

    let booksItem: Books = Books()

    booksItem._isbn = "1234"
    booksItem._category = "History"
    booksItem._author = "Harriet Tubman"
    booksItem._title = "My Life"

    //Save a new item
    dynamoDbObjectMapper.save(booksItem, completionHandler: {
        (error: Error?) -> Void in

        if let error = error {
            print("Amazon DynamoDB Save Error: \(error)")
            return
        }
        print("An item was saved.")
    })
}
```

Read (Load) an Item

Use the following code to read an item in your NoSQL Database table.

Android - Java

```
public void readBooks() {
    new Thread(new Runnable() {
        @Override
        public void run() {

            com.amazonaws.models.nosql.BooksDO booksItem = dynamoDBMapper.load(
                    com.amazonaws.models.nosql.BooksDO.class,
                    "ISBN1",       // Partition key (hash key)
                    "History");    // Sort key (range key)

            // Item read
             Log.d(LOG_TAG, String.format("Books Item: %s", booksItem.toString()));
        }
    }).start();
}
```

Android - Kotlin

```
fun readBooks() {
    thread(start = true) {
        val booksItem = ddbMapper.load(BooksDO::class.java,
            "ISBN1",    // Partition Key (hash key)
            "History")  // Sort key (range key)

        Log.d(LOG_TAG, "Books Item: $booksItem")
    }
}
```

iOS - Swift

```
func readBooks() {
   let dynamoDbObjectMapper = AWSDynamoDBObjectMapper.default()

   // Create data object using data model you created
   let booksItem: Books = Books();

   dynamoDbObjectMapper.load(
       Books.self,
       hashKey: "1234",
       rangeKey: "Harriet Tubman",
       completionHandler: {
           (objectModel: AWSDynamoDBObjectModel?, error: Error?) -> Void in
           if let error = error {
               print("Amazon DynamoDB Read Error: \(error)")
               return
           }
           print("An item was read.")
   })
}
```

Update an Item

Use the following code to update an item in your NoSQL Database table.

Android - Java

```
public void updateBooks() {
    final com.amazonaws.models.nosql.BooksDO booksItem = new
 com.amazonaws.models.nosql.BooksDO();


    booksItem.setIsbn("ISBN1");
    booksItem.setCategory("History");
    booksItem.setAuthor("Frederick M. Douglas");
    //  booksItem.setTitle("Escape from Slavery");

    new Thread(new Runnable() {
        @Override
        public void run() {


            // Using .save(bookItem) with no Title value makes that attribute value
 equal null
            // The .Savebehavior shown here leaves the existing value as is
            dynamoDBMapper.save(booksItem, new
 DynamoDBMapperConfig(DynamoDBMapperConfig.SaveBehavior.UPDATE_SKIP_NULL_ATTRIBUTES));

            // Item updated
```

```
        }
    }).start();
}
```

Android - Kotlin

```
fun updateBooks() {
    val booksItem = BooksDO().apply {
        isbn = "ISBN1"
        category = "History"
        author = "Frederick M. Douglas"
        // Do not set title - it will be removed from the item in DynamoDB
    }

    thread(start = true) {
        ddbMapper.save(booksItem,
 DynamoDBMapperConfig(DynamoDBMappConfig.SaveBehavior.UPDATE_SKIP_NULL_ATTERIBUTES))
    }
}
```

iOS - Swift

```
func updateBooks() {
   let dynamoDbObjectMapper = AWSDynamoDBObjectMapper.default()

   let booksItem: Books = Books()

   booksItem._isbn = "1234"
   booksItem._category = "History"
   booksItem._author = "Harriet Tubman"
   booksItem._title = "The Underground Railroad"


   dynamoDbObjectMapper.save(booksItem, completionHandler: {(error: Error?) -> Void in
       if let error = error {
           print(" Amazon DynamoDB Save Error: \(error)")
           return
       }
       print("An item was updated.")
   })
}
```

Delete an Item

Use the following code to delete an item in your NoSQL Database table.

Android - Java

```
public void deleteBooks() {
    new Thread(new Runnable() {
        @Override
        public void run() {

            com.amazonaws.models.nosql.BooksDO booksItem = new
 com.amazonaws.models.nosql.BooksDO();
            booksItem.setIsbn("ISBN1");       //partition key
            booksItem.setCategory("History"); //range key

            dynamoDBMapper.delete(booksItem);

            // Item deleted
```

```
        }
    }).start();
}
```

Android - Kotlin

```
fun deleteBook() {
    thread(start = true) {
        val booksItem = BooksDO().apply {
            isbn = "ISBN1"          // Partition key
            category = "History"    // Range key
        }
        ddbMapper.delete(booksItem)
    }
}
```

iOS - Swift

```
func deleteBooks() {
   let dynamoDbObjectMapper = AWSDynamoDBObjectMapper.default()

   let itemToDelete = Books()
   itemToDelete?._isbn = "1234"
   itemToDelete?._category = "History"

   dynamoDbObjectMapper.remove(itemToDelete!, completionHandler: {(error: Error?) ->
 Void in
       if let error = error {
           print(" Amazon DynamoDB Save Error: \(error)")
           return
       }
       print("An item was deleted.")
   })
}
```

Perform a Query

A query operation enables you to find items in a table. You must define a query using both the hash key (partition key) and range key (sort key) attributes of a table. You can filter the results by specifying the attributes you are looking for. For more information about `DynamoDBQueryExpression`, see the AWS Mobile SDK for Android API reference.

The following example code shows querying for books with partition key (hash key) `ISBN` and sort key (range key) Category beginning with `History`.

Android - Java

```
public void queryBook() {

     new Thread(new Runnable() {
         @Override
         public int hashCode() {
             return super.hashCode();
         }

         @Override
         public void run() {
             com.amazonaws.models.nosql.BooksDO book = new
 com.amazonaws.models.nosql.BooksDO();
             book.setIsbn("ISBN1");       //partition key
```

```
            book.setCategory("History"); //range key


            Condition rangeKeyCondition = new Condition()
                    .withComparisonOperator(ComparisonOperator.BEGINS_WITH)
                    .withAttributeValueList(new AttributeValue().withS("History"));
            DynamoDBQueryExpression queryExpression = new DynamoDBQueryExpression()
                    .withHashKeyValues(book)
                    .withRangeKeyCondition("Category", rangeKeyCondition)
                    .withConsistentRead(false);

            PaginatedList<BooksDO> result =
 dynamoDBMapper.query(com.amazonaws.models.nosql.BooksDO.class, queryExpression);

            Gson gson = new Gson();
            StringBuilder stringBuilder = new StringBuilder();

            // Loop through query results
            for (int i = 0; i < result.size(); i++) {
                String jsonFormOfItem = gson.toJson(result.get(i));
                stringBuilder.append(jsonFormOfItem + "\n\n");
            }

            // Add your code here to deal with the data result
            Log.d("Query results: ", stringBuilder.toString());

            if (result.isEmpty()) {
                // There were no items matching your query.
            }
         }
    }).start();
 }
```

Android - Kotlin

```
fun queryBooks() {
    thread(start = true) {
        val book = BooksDO().apply {
            isbn = "ISBN1"          // Partition key
            category = "History"    // Range key
        }

        val rangeKeyCondition = Condition()
            .withComparisonOperator(ComparisionOperator.BEGINS_WITH)
            .withAttrbiuteValueList(AttributeValue().withS("History"))
        val queryExpression = DynamoDBQUeryExpression()
            .withHashKeyValues(book)
            .withRangeKeyCondition("Category", rangeKeyCondition)
            .withConsistentRead(false)

        val result = ddbMapper.query(BooksDO::class.java, queryExpression) as
 PaginatedList<BooksDO>
        if (result.isEmpty()) {
            // There were no items matching your query
        } else {
            // loop through the result list and process the response
        }
    }
}
```

iOS - Swift

```
func queryBooks() {
```

```
   // 1) Configure the query
   let queryExpression = AWSDynamoDBQueryExpression()
    queryExpression.keyConditionExpression = "#isbn = :ISBN AND #category = :Category"

   queryExpression.expressionAttributeNames = [
       "#isbn": "ISBN",
       "#category": "Category"
   ]

   queryExpression.expressionAttributeValues = [
       ":ISBN" : "1234",
       ":Category" : "History"
   ]

   // 2) Make the query
   let dynamoDbObjectMapper = AWSDynamoDBObjectMapper.default()

   dynamoDbObjectMapper.query(Books.self, expression: queryExpression) { (output:
AWSDynamoDBPaginatedOutput?, error: Error?) in
       if error != nil {
           print("The request failed. Error: \(String(describing: error))")
       }
       if output != nil {
           for books in output!.items {
               let booksItem = books as? Books
               print("\(booksItem!._title!)")
           }
       }
   }
}
```

Next Steps

- To learn more about IAM policies, see Using IAM.
- To learn more about creating fine-grained access policies for Amazon DynamoDB, see DynamoDB on Mobile – Part 5: Fine-Grained Access Control.

iOS: Amazon DynamoDB Object Mapper API

Topics

Overview

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

The AWS Mobile SDK for iOS provides both low-level and high-level libraries for working with Amazon DynamoDB.

The high-level library described in this section provides Amazon DynamoDB object mapper which lets you map client-side classes to tables. Working within the data model defined on your client you can write simple, readable code that stores and retrieves objects in the cloud.

The dynamodb-low-level-client provides useful ways to perform operations like conditional writes and batch operations.

Setup

To set your project up to use the AWS SDK for iOS `dynamoDBObjectMapper`, take the following steps.

Setup the SDK, Credentials, and Services

To integrate `dynamoDBObjectMapper` into a new app, follow the steps described in Get Started to install the AWS Mobile SDK for iOS.

For apps that use an SDK version prior to 2.6.0, follow the steps on setup-options-for-aws-sdk-for-ios to install the AWS Mobile SDK for iOS. Then use the steps on cognito-auth-identity-for-ios-legacy to configure user credentials, and permissions.

Instantiate the Object Mapper API

In this section:

Topics

Import the AWSDynamoDB APIs

Add the following import statement to your project.

iOS - Swift

```
import AWSDynamoDB
```

iOS - Objective-C

```
#import <AWSDynamoDB/AWSDynamoDB.h>
```

Create Amazon DynamoDB Object Mapper Client

Use the AWSDynamoDBObjectMapper to map a client-side class to your database. The object mapper supports high-level operations like creating, getting, querying, updating, and deleting records. Create an object mapper as follows.

iOS - Swift

```
dynamoDBObjectMapper = AWSDynamoDBObjectMapper.default()
```

iOS - Objective-C

```
AWSDynamoDBObjectMapper *dynamoDBObjectMapper = [AWSDynamoDBObjectMapper
 defaultDynamoDBObjectMapper];
```

Object mapper methods return an `AWSTask` object. for more information, see Working with Asynchronous Tasks.

Define a Mapping Class

An Amazon DynamoDB database is a collection of tables, and a table can be described as follows:

- A table is a collection of items.
- Each item is a collection of attributes.
- Each attribute has a name and a value.

For the bookstore app, each item in the table represents a book, and each item has four attributes: *Title*, *Author*, *Price*, and *ISBN*.

Each item (Book) in the table has a **Primary key**, in this case, the primary key is `ISBN`.

To directly manipulate database items through their object representation, map each item in the Book table to a `Book` object in the client-side code, as shown in the following code. Attribute names are case sensitive.

iOS - Swift

```
import AWSDynamoDB

class Book : AWSDynamoDBObjectModel, AWSDynamoDBModeling  {
    @objc var Title:String?
    @objc var Author:String?
    @objc var Price:String?
    @objc var ISBN:String?

    class Amazon DynamoDBTableName() -> String {
        return "Books"
    }

    class func hashKeyAttribute() -> String {
        return "ISBN"
    }
}
```

iOS - Objective-C

```
#import <AWSDynamoDB/AWSDynamoDB.h>
#import "Book.h"

@interface Book : AWSDynamoDBObjectModel <AWSDynamoDBModeling>

@property (nonatomic, strong) NSString *Title;
@property (nonatomic, strong) NSString *Author;
@property (nonatomic, strong) NSNumber *Price;
@property (nonatomic, strong) NSString *ISBN;

@end


@implementation Book

+ (NSString *)dynamoDBTableName {
    return @"Books";
}
```

```
+ (NSString *)hashKeyAttribute {
    return @"ISBN";
}

@end
```

Note
As of SDK version 2.0.16, the `AWSDynamoDBModel` mapping class is deprecated and replaced by `AWSDynamoDBObjectModel`. For information on migrating your legacy code, see awsdynamodb-model.

To conform to the `AWSDynamoDBModeling` protocol, implement `dynamoDBTableName`, which returns the name of the table, and `hashKeyAttribute`, which returns the name of the primary key. If the table has a range key, implement `+ (NSString *)rangeKeyAttribute`.

CRUD Operations

In this section:

Topics

The Amazon DynamoDB table, mapping class, and object mapper client enable your app to interact with objects in the cloud.

Save an Item

The save: method saves an object to Amazon DynamoDB, using the default configuration. As a parameter, `save:` takes a an object that inherits from `AWSDynamoDBObjectModel` and conforms to the `AWSDynamoDBModeling` protocol. The properties of this object will be mapped to attributes in Amazon DynamoDB table.

To create the object to be saved take the following steps.

1. Define the object and it's properties to match your table model.

   iOS - Swift

   ```
   let myBook = Book()
   myBook?.ISBN = "3456789012"
   myBook?.Title = "The Scarlet Letter"
   myBook?.Author = "Nathaniel Hawthorne"
   myBook?.Price = 899 as NSNumber?
   ```

   iOS - Objective-C

   ```
   Book *myBook = [Book new];
   myBook.ISBN = @"3456789012";
   myBook.Title = @"The Scarlet Letter";
   myBook.Author = @"Nathaniel Hawthorne";
   myBook.Price = [NSNumber numberWithInt:899];
   ```

2. Pass the object to the `save:` method.

iOS - Swift

```
dynamoDBObjectMapper.save(myBook).continueWith(block: { (task:AWSTask<AnyObject>!) ->
 Any? in
     if let error = task.error as? NSError {
         print("The request failed. Error: \(error)")
     } else {
         // Do something with task.result or perform other operations.
     }
 })
```

iOS - Objective-C

```
[[dynamoDBObjectMapper save:myBook]
continueWithBlock:^id(AWSTask *task) {
     if (task.error) {
         NSLog(@"The request failed. Error: [%@]", task.error);
     } else {
         //Do something with task.result or perform other operations.
     }
     return nil;
 }];
```

Save Behavior Options

The AWS Mobile SDK for iOS supports the following save behavior options:

- `AWSDynamoDBObjectMapperSaveBehaviorUpdate`

  This option does not affect unmodeled attributes on a save operation. Passing a nil value for the modeled attribute removes the attribute from the corresponding item in Amazon DynamoDB. By default, the object mapper uses this behavior.
- `AWSDynamoDBObjectMapperSaveBehaviorUpdateSkipNullAttributes`

  This option is similar to the default update behavior, except that it ignores any null value attribute(s) and does not remove them from an item in Amazon DynamoDB.
- `AWSDynamoDBObjectMapperSaveBehaviorAppendSet`

  This option treats scalar attributes (String, Number, Binary) the same as the `AWSDynamoDBObjectMapperSaveBehaviorUpdateSkipNullAttributes` option. However, for set attributes, this option appends to the existing attribute value instead of overriding it. The caller must ensure that the modeled attribute type matches the existing set type; otherwise, a service exception occurs.
- `AWSDynamoDBObjectMapperSaveBehaviorClobber`

  This option clears and replaces all attributes, including unmodeled ones, on save. Versioned field constraints are be disregarded.

The following code provides an example of setting a default save behavior on the object mapper.

iOS - Swift

```
let updateMapperConfig = AWSDynamoDBObjectMapperConfiguration()
updateMapperConfig.saveBehavior = .updateSkipNullAttributes
```

iOS - Objective-C

```
AWSDynamoDBObjectMapperConfiguration *updateMapperConfig =
 [AWSDynamoDBObjectMapperConfiguration new];
updateMapperConfig.saveBehavior =
 AWSDynamoDBObjectMapperSaveBehaviorUpdateSkipNullAttributes;
```

Use `updateMapperConfig` as an argument when calling save:configuration:.

Retrieve an Item

Using an object's primary key, in this case, `ISBN`, we can load the corresponding item from the database. The following code returns the Book item with an ISBN of `6543210987`.

iOS - Swift

```
dynamoDBObjectMapper.load(Book.self, hashKey: "6543210987"
 rangeKey:nil).continueWith(block: { (task:AWSTask<AnyObject>!) -> Any? in
     if let error = task.error as? NSError {
         print("The request failed. Error: \(error)")
     } else if let resultBook = task.result as? Book {
         // Do something with task.result.
     }
     return nil
 })
```

iOS - Objective-C

```
[[dynamoDBObjectMapper load:[Book class] hashKey:@"6543210987" rangeKey:nil]
continueWithBlock:^id(AWSTask *task) {
    if (task.error) {
        NSLog(@"The request failed. Error: [%@]", task.error);
    } else {
        //Do something with task.result.
    }
    return nil;
}];
```

The object mapper creates a mapping between the `Book` item returned from the database and the `Book` object on the client (here, `resultBook`). Access the title at `resultBook.Title`.

Since the Books database does not have a range key, `nil` was passed to the `rangeKey` parameter.

Update an Item

To update an item in the database, just set new attributes and save the objects. The primary key of an existing item, `myBook.ISBN` in the `Book` object mapper example, cannot be changed. If you save an existing object with a new primary key, a new item with the same attributes and the new primary key are created.

Delete an Item

To delete a table row, use the *remove:* method.

iOS - Swift

```
 let bookToDelete = Book()
```

```
 bookToDelete?.ISBN = "4456789012";

dynamoDBObjectMapper.remove(bookToDelete).continueWith(block:
 { (task:AWSTask<AnyObject>!) -> Any? in
     if let error = task.error as? NSError {
         print("The request failed. Error: \(error)")
     } else {
         // Item deleted.
     }
 })
```

iOS - Objective-C

```
Book *bookToDelete = [Book new];
bookToDelete.ISBN = @"4456789012";

[[dynamoDBObjectMapper remove:bookToDelete]
 continueWithBlock:^id(AWSTask *task) {

     if (task.error) {
         NSLog(@"The request failed. Error: [%@]", task.error);
     } else {
         //Item deleted.
     }
     return nil;
 }];
```

Perform a Scan

A scan operation retrieves in an undetermined order.

The `scan:expression:` method takes two parameters: the class of the resulting object and an instance of `AWSDynamoDBScanExpression`, which provides options for filtering results.

The following example shows how to create an `AWSDynamoDBScanExpression` object, set its `limit` property, and then pass the `Book` class and the expression object to `scan:expression:`.

iOS - Swift

```
 let scanExpression = AWSDynamoDBScanExpression()
 scanExpression.limit = 20

dynamoDBObjectMapper.scan(Book.self, expression: scanExpression).continueWith(block:
 { (task:AWSTask<AnyObject>!) -> Any? in
     if let error = task.error as? NSError {
         print("The request failed. Error: \(error)")
     } else if let paginatedOutput = task.result {
         for book in paginatedOutput.items as! Book {
             // Do something with book.
         }
     }
 })
```

iOS - Objective-C

```
AWSDynamoDBScanExpression *scanExpression = [AWSDynamoDBScanExpression new];
scanExpression.limit = @10;

[[dynamoDBObjectMapper scan:[Book class]
        expression:scanExpression]
```

```
continueWithBlock:^id(AWSTask *task) {
     if (task.error) {
         NSLog(@"The request failed. Error: [%@]", task.error);
     } else {
         AWSDynamoDBPaginatedOutput *paginatedOutput = task.result;
         for (Book *book in paginatedOutput.items) {
             //Do something with book.
         }
     }
     return nil;
}];
```

Filter a Scan ~~~~~~~~~~~~-

The output of a scan is returned as an `AWSDynamoDBPaginatedOutput` object. The array of returned items is in the `items` property.

The `scanExpression` method provides several optional parameters. Use `filterExpression` and `expressionAttributeValues` to specify a scan result for the attribute names and conditions you define. For more information about the parameters and the API, see AWSDynamoDBScanExpression.

The following code scans the Books table to find books with a price less than 50.

iOS - Swift

```
 let scanExpression = AWSDynamoDBScanExpression()
 scanExpression.limit = 10
 scanExpression.filterExpression = "Price < :val"
 scanExpression.expressionAttributeValues = [":val": 50]

dynamoDBObjectMapper.scan(Book.self, expression: scanExpression).continueWith(block:
 { (task:AWSTask<AnyObject>!) -> Any? in
   if let error = task.error as? NSError {
       print("The request failed. Error: \(error)")
   } else if let paginatedOutput = task.result {
       for book in paginatedOutput.items as! Book {
           // Do something with book.
       }
   }
 })
```

iOS - Objective-C

```
AWSDynamoDBScanExpression *scanExpression = [AWSDynamoDBScanExpression new];
scanExpression.limit = @10;
scanExpression.filterExpression = @"Price < :val";
scanExpression.expressionAttributeValues = @{@":val":@50};

[[dynamoDBObjectMapper scan:[Book class]
             expression:scanExpression]
continueWithBlock:^id(AWSTask *task) {
     if (task.error) {
         NSLog(@"The request failed. Error: [%@]", task.error);
     } else {
         AWSDynamoDBPaginatedOutput *paginatedOutput = task.result;
         for (Book *book in paginatedOutput.items) {
             //Do something with book.
         }
     }
     return nil;
```

```
}];
```

You can also use the `projectionExpression` property to specify the attributes to retrieve from the ``Books table. For example adding `scanExpression.projectionExpression = @"ISBN, Title, Price";` in the previous code snippet retrieves only those three properties in the book object. The `Author` property in the book object will always be nil.

Perform a Query

The query API enables you to query a table or a secondary index. The `query:expression:` method takes two parameters: the class of the resulting object and an instance of `AWSDynamoDBQueryExpression`.

To query an index, you must also specify the `indexName`. You must specify the `hashKeyAttribute` if you query a global secondary with a different `hashKey`. If the table or index has a range key, you can optionally refine the results by providing a range key value and a condition.

The following example illustrates querying the *Books* index table to find all books whose author is "John Smith", with a price less than 50.

iOS - Swift

```
let queryExpression = AWSDynamoDBQueryExpression()
queryExpression.indexName = "Author-Price-index"

queryExpression.keyConditionExpression = @"Author = :authorName AND Price < :val";
queryExpression.expressionAttributeValues = @{@":authorName": @"John Smith", @":val":
@50};

dynamoDBObjectMapper.query(Book.self, expression: queryExpression).continueWith(block:
{ (task:AWSTask<AnyObject>!) -> Any? in
    if let error = task.error as? NSError {
          print("The request failed. Error: \(error)")
    } else if let paginatedOutput = task.result {
        for book in paginateOutput.items as! Book {
            // Do something with book.
        }
    }
    return nil
})
```

iOS - Objective-C

```
AWSDynamoDBQueryExpression *queryExpression = [AWSDynamoDBQueryExpression new];

queryExpression.indexName = @"Author-Price-index";

queryExpression.keyConditionExpression = @"Author = :authorName AND Price < :val";

queryExpression.expressionAttributeValues = @{@":authorName": @"John Smith",
 @":val":@50};

[[dynamoDBObjectMapper query:[Book class]
        expression:queryExpression]
continueWithBlock:^id(AWSTask *task) {
    if (task.error) {
        NSLog(@"The request failed. Error: [%@]", task.error);
    } else {
        AWSDynamoDBPaginatedOutput *paginatedOutput = task.result;
```

```
        for (Book *book in paginatedOutput.items) {
            //Do something with book.
        }
    }
    return nil;
}];
```

In the preceding example, `indexName` is specified to demonstrate querying an index. The query expression is specified using `keyConditionExpression` and the values used in the expression using `expressionAttributeValues`.

You can also provide `filterExpression` and `projectionExpression` in `AWSDynamoDBQueryExpression`. The syntax is the same as that used in a scan operation.

For more information, see AWSDynamoDBQueryExpression.

Migrating AWSDynamoDBModel to AWSDynamoDBObjectModel

As of SDK version 2.0.16, the `AWSDynamoDBModel` mapping class is deprecated and replaced by `AWSDynamoDBObjectModel`.The deprecated `AWSDynamoDBModel` used *NSArray* to represent multi-valued types (`String Set`, `Number Set`, and `Binary Set`); it did not support `Boolean`, `Map`, or `List` types. The new `AWSDynamoDBObjectModel` uses `NSSet` for multi-valued types and supports `Boolean`, `Map`, and `List`. For the `Boolean` type, you create an `NSNumber` using `[NSNumber numberWithBool:YES]` or using the shortcuts `@YES` and `@NO`. For the Map type, create using `NSDictionary`. For the List type, create using `NSArray`.

Additional Resources

- Amazon DynamoDB Developer Guide
- Amazon DynamoDB API Reference

iOS: Amazon DynamoDB Low-level Client

Topics

Overview

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

The AWS Mobile SDK for iOS provides both low-level and high-level libraries for working Amazon DynamoDB.

The low-level client described in this section allows the kind of direct access to Amazon DynamoDB tables useful for NoSQL and other non-relational data designs. The low-level client also supports conditional data writes to mitigate simultaneous write conflicts and batch data writes.

The high-level library includes dynamodb-object-mapper, which lets you map client-side classes to access and manipulate Amazon Dynamo tables.

Setup

To set your project up to use the AWS SDK for iOS TransferUtility, take the following steps.

1. Setup the SDK, Credentials, and Services

To use the low-level DynamoDB mobile client in a new app, follow the steps described in Get Started to install the AWS Mobile SDK for iOS.

For apps that use an SDK version prior to 2.6.0, follow the steps on setup-options-for-aws-sdk-for-ios to install the AWS Mobile SDK for iOS. Then use the steps on cognito-auth-identity-for-ios-legacy to configure user credentials, and permissions.

2. Create or Use an Existing Amazon DynamoDB Table

Follow the steps on <dynamodb-setup-for-ios-legacy> to create a table.

3. Import the AWSDynamoDB APIs

Add the following import statement to your project.

iOS - Swift

```
import AWSDynamoDB
```

iOS - Objective-C

```
#import <AWSDynamoDB/AWSDynamoDB.h>
```

Conditional Writes Using the Low-Level Client

In a multi-user environment, multiple clients can access the same item and attempt to modify its attribute values at the same time. To help clients coordinate writes to data items, the Amazon DynamoDB low-level client supports conditional writes for *PutItem*, *DeleteItem*, and *UpdateItem* operations. With a conditional write, an operation succeeds only if the item attributes meet one or more expected conditions; otherwise, it returns an error.

In the following example, we update the price of an item in the `Books` table if the `Price` attribute of the item has a value of `999`.

iOS - Swift

```
Amazon DynamoDB = AWSDynamoDB.default()
let updateInput = AWSDynamoDBUpdateItemInput()

let hashKeyValue = AWSDynamoDBAttributeValue()
hashKeyValue?.s = "4567890123"

updateInput?.tableName = "Books"
updateInput?.key = ["ISBN": hashKeyValue!]

let oldPrice = AWSDynamoDBAttributeValue()
oldPrice?.n = "999"

let expectedValue = AWSDynamoDBExpectedAttributeValue()
expectedValue?.value = oldPrice

let newPrice = AWSDynamoDBAttributeValue()
```

```
 newPrice?.n = "1199"

 let valueUpdate = AWSDynamoDBAttributeValueUpdate()
 valueUpdate?.value = newPrice
 valueUpdate?.action = .put

 updateInput?.attributeUpdates = ["Price": valueUpdate!]
 updateInput?.expected = ["Price": expectedValue!]
 updateInput?.returnValues = .updatedNew

Amazon DynamoDB.updateItem(updateInput!).continueWith
 { (task:AWSTask<AWSDynamoDBUpdateItemOutput>) -> Any? in
     if let error = task.error as? NSError {
         print("The request failed. Error: \(error)")
         return nil
     }

     // Do something with task.result

     return nil
 }
```

iOS - Objective-C

```
AWSDynamoDB *dynamoDB = [AWSDynamoDB defaultDynamoDB];
AWSDynamoDBUpdateItemInput *updateInput = [AWSDynamoDBUpdateItemInput new];

AWSDynamoDBAttributeValue *hashKeyValue = [AWSDynamoDBAttributeValue new];
hashKeyValue.S = @"4567890123";

updateInput.tableName = @"Books";
updateInput.key = @{ @"ISBN" : hashKeyValue };

AWSDynamoDBAttributeValue *oldPrice = [AWSDynamoDBAttributeValue new];
oldPrice.N = @"999";

AWSDynamoDBExpectedAttributeValue *expectedValue = [AWSDynamoDBExpectedAttributeValue
 new];
expectedValue.value = oldPrice;

AWSDynamoDBAttributeValue *newPrice = [AWSDynamoDBAttributeValue new];
newPrice.N = @"1199";

AWSDynamoDBAttributeValueUpdate *valueUpdate = [AWSDynamoDBAttributeValueUpdate new];
valueUpdate.value = newPrice;
valueUpdate.action = AWSDynamoDBAttributeActionPut;

updateInput.attributeUpdates = @{@"Price": valueUpdate};
updateInput.expected = @{@"Price": expectedValue};
updateInput.returnValues = AWSDynamoDBReturnValueUpdatedNew;

[[dynamoDB updateItem:updateInput]
 continueWithBlock:^id(AWSTask *task) {
     if (task.error) {
         NSLog(@"The request failed. Error: [%@]", task.error);
     } else {
         //Do something with task.result.
     }
     return nil;
 }];
```

Conditional writes are idempotent. In other words, if a conditional write request is made multiple times, the update will be performed only in the first instance unless the content of the request changes. In the

preceding example, sending the same request a second time results in a *ConditionalCheckFailedException*, because the expected condition is not met after the first update.

Batch Operations Using the Low-Level Client

The Amazon DynamoDB low-level client provides batch write operations to put items in the database and delete items from the database. You can also use batch get operations to return the attributes of one or more items from one or more tables.

The following example shows a batch write operation.

iOS - Swift

```
 Amazon DynamoDB = AWSDynamoDB.default()

 //Write Request 1
 let hashValue1 = AWSDynamoDBAttributeValue()
 hashValue1?.s = "3210987654"
 let otherValue1 = AWSDynamoDBAttributeValue()
 otherValue1?.s = "Some Title"

 let writeRequest = AWSDynamoDBWriteRequest()
 writeRequest?.putRequest = AWSDynamoDBPutRequest()
 writeRequest?.putRequest?.item = ["ISBN": hashValue1!, "Title": otherValue1!]

 //Write Request 2
 let hashValue2 = AWSDynamoDBAttributeValue()
 hashValue2?.s = "8901234567"
 let otherValue2 = AWSDynamoDBAttributeValue()
 otherValue2?.s = "Another Title"

 let writeRequest2 = AWSDynamoDBWriteRequest()
 writeRequest2?.putRequest = AWSDynamoDBPutRequest()
 writeRequest2?.putRequest?.item = ["ISBN": hashValue2!, "Title": otherValue2!]

 let batchWriteItemInput = AWSDynamoDBBatchWriteItemInput()
 batchWriteItemInput?.requestItems = ["Books": [writeRequest!, writeRequest2!]]

Amazon DynamoDB.batchWriteItem(batchWriteItemInput!).continueWith
 { (task:AWSTask<AWSDynamoDBBatchWriteItemOutput>) -> Any? in
     if let error = task.error as? NSError {
         print("The request failed. Error: \(error)")
         return nil
     }

     // Do something with task.result

     return nil
 }
```

iOS - Objective-C

```
AWSDynamoDB *dynamoDB = [AWSDynamoDB defaultDynamoDB];

//Write Request 1
AWSDynamoDBAttributeValue *hashValue1 = [AWSDynamoDBAttributeValue new];
hashValue1.S = @"3210987654";
AWSDynamoDBAttributeValue *otherValue1 = [AWSDynamoDBAttributeValue new];
otherValue1.S = @"Some Title";

AWSDynamoDBWriteRequest *writeRequest = [AWSDynamoDBWriteRequest new];
writeRequest.putRequest = [AWSDynamoDBPutRequest new];
writeRequest.putRequest.item = @{
```

```
                                @"ISBN" : hashValue1,
                                @"Title" : otherValue1
                                };

//Write Request 2
AWSDynamoDBAttributeValue *hashValue2 = [AWSDynamoDBAttributeValue new];
hashValue2.S = @"8901234567";
AWSDynamoDBAttributeValue *otherValue2 = [AWSDynamoDBAttributeValue new];
otherValue2.S = @"Another Title";

AWSDynamoDBWriteRequest *writeRequest2 = [AWSDynamoDBWriteRequest new];
writeRequest2.putRequest = [AWSDynamoDBPutRequest new];
writeRequest2.putRequest.item = @{
                                @"ISBN" : hashValue2,
                                @"Title" : otherValue2
                                };

AWSDynamoDBBatchWriteItemInput *batchWriteItemInput = [AWSDynamoDBBatchWriteItemInput
 new];
batchWriteItemInput.requestItems = @{@"Books": @[writeRequest,writeRequest2]};

[[dynamoDB batchWriteItem:batchWriteItemInput]
 continueWithBlock:^id(AWSTask *task) {
     if (task.error) {
         NSLog(@"The request failed. Error: [%@]", task.error);
     } else  {
        //Do something with task.result.
    }
    return nil;
}];
```

Additional Resources

- Amazon DynamoDB Developer Guide
- Amazon DynamoDB API Reference

How To: Serverless Code with AWS Lambda

Just Getting Started?	Use streamlined steps (p. 75) to install the SDK and integrate AWS Lambda.

Or, use the contents of this page if your app will integrate existing AWS services.

This section provides information on the steps for achieving specific tasks for integrating your AWS Lambda functions into your Android and iOS apps.

Topics

- Android: Execute Code On Demand with AWS Lambda (p. 261)
- iOS: Execute Code On Demand with AWS Lambda (p. 266)

Android: Execute Code On Demand with AWS Lambda

Just Getting Started?	Use streamlined steps (p. 75) to install the SDK and integrate features.

Or, use the contents of this page if your app will integrate existing AWS services.

Overview

AWS Lambda is a compute service that runs your code in response to events and automatically manages the compute resources for you, making it easy to build applications that respond quickly to new information. The AWS Mobile SDK for Android enables you to call Lambda functions from your Android mobile apps.

The tutorial below explains how to integrate AWS Lambda with your app.

Setup

Prerequisites

You must complete all of the instructions on the Android: Setup Options for the SDK (p. 133) page before beginning this tutorial.

Create a Lambda Function in the AWS Console

For this tutorial, let's use a simple "echo" function that returns the input. Follow the steps described at AWS Lambda Getting Started, replacing the function code with the code below:

```
exports.handler = function(event, context) {
     console.log("Received event");
     context.succeed("Hello "+ event.firstName + "using " +
 context.clientContext.deviceManufacturer);
  }
```

Set IAM Permissions

The default IAM role policy grants your users access to Amazon Mobile Analytics and Amazon Cognito Sync. To use AWS Lambda in your application, you must configure the IAM role policy so that it allows your application and your users access to AWS Lambda. The IAM policy in the following steps allows the user to perform the actions shown in this tutorial on a given AWS Lambda function identified by its Amazon Resource Name (ARN). To find the ARN go to the Lambda Console and click the **Function name**.

To set IAM Permissions for AWS Lambda:

1. Navigate to the IAM Console and click **Roles** in the left-hand pane.
2. Type your identity pool name into the search box. Two roles will be listed: one for unauthenticated users and one for authenticated users.
3. Click the role for unauthenticated users (it will have `unauth` appended to your Identity Pool name).
4. Click the **Create Role Policy** button, select **Custom Policy**, and then click the **Select** button.
5. Enter a name for your policy and paste in the following policy document, replacing the function's `Resource` value with the ARN for your function (click your function's **Function name** in the AWS Lambda console to view its ARN).

```
{
   "Statement": [{
      "Effect": "Allow",
      "Action": [
          "lambda:invokefunction"
      ],
      "Resource": [
         "arn:aws:lambda:us-west-2:012345678901:function:yourFunctionName"
      ]
```

```
    }]
}
```

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

To learn more about IAM policies, see IAM documentation.

Set Permissions in Your Android Manifest

In your `AndroidManifest.xml`, add the following permission

```
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

Initialize LambdaInvokerFactory

Android - Java

Pass your initialized Amazon Cognito credentials provider to the `LambdaInvokerFactory` constructor:

```
LambdaInvokerFactory factory = new LambdaInvokerFactory(
    myActivity.getApplicationContext(),
    REGION,
    credentialsProvider);
```

Android - Kotlin

Pass your initialized Amazon Cognito credentials provider to the `LambdaInvokerFactory` constructor:

```
val factory = LambdaInvokerFactory(applicationContext,
    REGION, credentialsProvider)
```

Declare Data Types

Create a Lambda proxy

Declare an interface containing one method for each Lambda function call. Each method in the interface must be decorated with the "@LambdaFunction" annotation. The LambdaFunction attribute can take 3 optional parameters:

- `functionName` allows you to specify the name of the Lambda function to call when the method is executed, by default the name of the method is used.
- `logType` is valid only when invocationType is set to "Event". If set, AWS Lambda will return the last 4KB of log data produced by your Lambda Function in the x-amz-log-results header.
- `invocationType` specifies how the Lambda function will be invoked. Can be one of the following values:
  - Event: calls the Lambda Function asynchronously
  - RequestResponse: calls the Lambda Function synchronously
  - DryRun: allows you to validate access to a Lambda Function without executing it

The following code shows how to create a Lambda proxy:

Android - Java

```
package com.amazonaws.demo.lambdainvoker;

import com.amazonaws.mobileconnectors.lambdainvoker.LambdaFunction;

public interface MyInterface {
   /**
    * Invoke lambda function "echo". The function name is the method name
    */
   @LambdaFunction
   String echo(NameInfo nameInfo)

   /**
    * Invoke lambda function "echo". The functionName in the annotation
    * overrides the default which is the method name
    */
   @LambdaFunction(functionName = "echo")
   void noEcho(NameInfo nameInfo)
}
```

Android - Kotlin

```
package com.amazonaws.demo.lambdainvoker;

import com.amazonaws.mobileconnectors.lambdainvoker.LambdaFunction;

interface MyInterface {
   /**
    * Invoke lambda function "echo". The function name is the method name
    */
   @LambdaFunction
   fun echo(nameInfo: NameInfo): String

   /**
    * Invoke lambda function "echo". The functionName in the annotation
    * overrides the default which is the method name
    */
   @LambdaFunction(functionName = "echo")
   fun noEcho(nameInfo: NameInfo): Unit
}
```

Invoke the Lambda Function

Note
Do not invoke the Lambda function from the main thread as it results in a network call.

The following code shows how to initialize the Cognito Caching Credentials Provider and invoke a Lambda function. The value for `IDENTITY_POOL_ID` will be specific to your account. Ensure the region is the same as the Lambda function you are trying to invoke.

Android - Java

```
// Create an instance of CognitoCachingCredentialsProvider
CognitoCachingCredentialsProvider credentialsProvider =
    new CognitoCachingCredentialsProvider(
       myActivity.getApplicationContext(),
       IDENTITY_POOL_ID,
       Regions.YOUR_REGION);
```

```
// Create a LambdaInvokerFactory, to be used to instantiate the Lambda proxy
LambdaInvokerFactory factory = new LambdaInvokerFactory(
   myActivity.getApplicationContext(),
   REGION,
   credentialsProvider);

// Create the Lambda proxy object with default Json data binder.
// You can provide your own data binder by implementing
// LambdaDataBinder
MyInterface myInterface = factory.build(MyInterface.class);

// Create an instance of the POJO to transfer data
NameInfo nameInfo = new NameInfo("John", "Doe");

// The Lambda function invocation results in a network call
// Make sure it is not called from the main thread
new AsyncTask<NameInfo, Void, String>() {
    @Override
    protected String doInBackground(NameInfo... params) {
        // invoke "echo" method. In case it fails, it will throw a
        // LambdaFunctionException.
        try {
            return myInterface.echo(params[0]);
        } catch (LambdaFunctionException lfe) {
            Log.e(TAG, "Failed to invoke echo", lfe);
            return null;
        }
    }

    @Override
    protected void onPostExecute(String result) {
        if (result == null) {
             return;
        }

        // Do a toast
        Toast.makeText(MainActivity.this, result, Toast.LENGTH_LONG).show();
    }
}.execute(nameInfo);
```

Android - Kotlin

```
// Create an instance of CognitoCachingCredentialsProvider
val credentialsProvider = CognitoCachingCredentialsProvider(
   this@MainActivity.applicationContext,
   IDENTITY_POOL_ID,
   Regions.IDENTITY_POOL_REGION)

// Create a LambdaInvokerFactory, to be used to instantiate the Lambda proxy
val factory = LambdaInvokerFactory(
   this@MainActivity.applicationContext,
   LAMBDA_REGION,
   credentialsProvider)

// Create the Lambda proxy object with default Json data binder.
// You can provide your own data binder by implementing
// LambdaDataBinder
val  myInterface = factory.build(MyInterface::class.java);

// Create an instance of the POJO to transfer data
val nameInfo = NameInfo("John", "Doe");

// The Lambda function invocation results in a network call
// Make sure it is not called from the main thread
```

```
thread(start = true) {
   // Invoke "echo" method.  In case it fails, it will throw an exception
   try {
       val response: String = myInterface.echo(nameInfo)
       runOnUiThread {
           Toast.makeText(this@MainActivity, result, Toast.LENGTH_LONG).show()
       }
   } catch (ex: LambdaFunctionException) {
       Log.e(TAG, "Lambda execution failed")
   }
}
```

Now whenever the Lambda function is invoked, you should see an application toast with the text "Hello John using <device>".

To get started using streamlined steps for setting up and using lambda functions to handle cloud API calls, see Add AWS Mobile Cloud Logic (p. 75).

iOS: Execute Code On Demand with AWS Lambda

Just Getting Started?	Use streamlined steps (p. 75) to install the SDK and integrate AWS Lambda functions.

Or, use the contents of this page if your app will integrate existing AWS services.

Topics

Overview

The AWS Lambda service makes it easy to create scalable, secure, and highly available backends for your mobile apps without the need to provision or manage infrastructure.

You can create secure logical functions in the cloud that can be called directly from your iOS app. Your AWS Lambda code, written in C#, Node.js, Python, or Java, can implement standalone logic, extend your app to a range of AWS services, and/or connect to services and applications external to AWS.

The availability and cost of a AWS Lambda function automatically scales to amount of traffic it receives. Functions can also be accessed from an iOS app through Amazon API Gateway, giving features like global provisioning, enterprise grade monitoring, throttling and control of access.

Setup

This section provides a step-by-step guide for getting started with AWS Lambda using the AWS Mobile SDK for iOS.

1. Install the SDK

   Add the AWS SDK for iOS to your project and import the APIs you need, by following the steps described in Set Up the AWS SDK for iOS.
2. Configure Credentials

To use Amazon Cognito to create AWS identities and credentials that give your users access to your app's AWS resources, follow the steps described at :ref:`Amazon Cognito Setup for iOS <cognito-auth-aws-identity-for-ios>`.

3. Create and Configure a Lamda Function
   a. Sign in to the AWS Lambda console.
   b. Choose **Create a Lamda function**.
   c. Choose the **Blank Function** template.

      Note that dozens of function templates that connect to other AWS services are available.
   d. Choose **Next**.

      Note that the console allows you to configure triggers for a function from other AWS services, these won't be used in this walkthrough.
   e. Type a **Name** and select *Node.js* as the **Runtime** language.
   f. Under **Lambda function handler and role**, select **Create new role from template(s)**. Type a **Role name**. Select the **Policy template** named **Simple Microservice permissions**.
   g. Choose **Next**.
   h. Choose **Create function**.

Invoking an AWS Lambda Function

The SDK enables you to call AWS Lambda functions from your iOS mobile apps, using the AWSLambdaInvoker class. When invoked from this SDK, AWS Lambda functions receive data about the device and the end user identity through client and identity context objects. To learn more about using these contexts to create rich, and personalized app experiences, see Client Context (p. 270) and Identity Context (p. 271).

Import AWS Lambda API

To use the *lambdainvoker* API, use the following import statement:

iOS - Swift

```
import AWSLambda
```

Objective C

```
#import <AWSLambda/AWSLambda.h>
```

Call lambdaInvoker

`AWSLambdaInvoker` provides a high-level abstraction for AWS Lambda. When `invokeFunction JSONObject` is invoked, the JSON object is serialized into JSON data and sent to the AWS Lambda service. AWS Lambda returns a JSON encoded response that is deserialized into a JSON object.

A valid JSON object must have the following properties:

- All objects are instances of string, number, array, dictionary or null objects.
- All dictionary keys are instances of string objects.
- Numbers are not `NaN` or `infinity`.

The following is an example of valid request.

iOS - Swift

```
let lambdaInvoker = AWSLambdaInvoker.default()
let jsonObject: [String: Any] = ["key1" : "value1",
                         "key2" : 2 ,
                         "key3" : [1, 2],
                         "isError" : false]

lambdaInvoker.invokeFunction("myFunction", jsonObject: jsonObject)
    .continueWith(block: {(task:AWSTask<AnyObject>) -> Any? in
    if( task.error != nil) {
        print("Error: \(task.error!)")
        return nil
    }

    // Handle response in task.result
    return nil
})
```

Objective C

```
AWSLambdaInvoker *lambdaInvoker = [AWSLambdaInvoker defaultLambdaInvoker];

[[lambdaInvoker invokeFunction:@"myFunction"
            JSONObject:@{@"key1" : @"value1",
                         @"key2" : @2,
                         @"key3" : [NSNull null],
                         @"key4" : @[@1, @"2"],
                         @"isError" : @NO}] continueWithBlock:^id(AWSTask *task) {
    // Handle response
    return nil;
}];
```

Using function returns

On successful execution, *task.result* contains a JSON object. For instance, if *myFunctions* returns a dictionary, you can cast the result to a dictionary object as follows.

iOS - Swift

```
if let JSONDictionary = task.result as? NSDictionary {
    print("Result: \(JSONDictionary)")
    print("resultKey: \(JSONDictionary["resultKey"])")
}
```

Objective C

```
if (task.result) {
    NSLog(@"Result: %@", task.result);
    NSDictionary *JSONObject = task.result;
    NSLog(@"result: %@", JSONObject[@"resultKey"]);
}
```

Handling service execution errors

On failed AWS Lambda service execution, *task.error* may contain a *NSError* with *AWSLambdaErrorDomain* domain and the following error code.

- *AWSLambdaErrorUnknown*

- *AWSLambdaErrorService*
- *AWSLambdaErrorResourceNotFound*
- *AWSLambdaErrorInvalidParameterValue*

On failed function execution, *task.error* may contain a *NSError* with *AWSLambdaInvokerErrorDomain* domain and the following error code:

- *AWSLambdaInvokerErrorTypeUnknown*
- *AWSLambdaInvokerErrorTypeFunctionError*

When *AWSLambdaInvokerErrorTypeFunctionError* error code is returned, *error.userInfo* may contain a function error from your AWS Lambda function with *AWSLambdaInvokerFunctionErrorKey* key.

The following code shows error handling.

iOS - Swift

```
if let error = task.error as? NSError {
    if (error.domain == AWSLambdaInvokerErrorDomain) &&
 (AWSLambdaInvokerErrorType.functionError == AWSLambdaInvokerErrorType(rawValue:
 error.code)) {
        print("Function error: \(error.userInfo[AWSLambdaInvokerFunctionErrorKey])")
    } else {
        print("Error: \(error)")
    }
    return nil
}
```

Objective C

```
if (task.error) {
    NSLog(@"Error: %@", task.error);
    if ([task.error.domain isEqualToString:AWSLambdaInvokerErrorDomain]
        && task.error.code == AWSLambdaInvokerErrorTypeFunctionError) {
        NSLog(@"Function error: %@",
 task.error.userInfo[AWSLambdaInvokerFunctionErrorKey]);
    }
}
```

Comprehensive example

The following code shows invoking an AWS Lambda call and handling returns and errors all together.

iOS - Swift

```
let lambdaInvoker = AWSLambdaInvoker.default()

let jsonObject: [String: Any] = ["key1" : "value1",
                      "key2" : 2,
                      "key3" : [1, 2],
                      "isError" : false]

lambdaInvoker.invokeFunction("myFunction", jsonObject: jsonObject).continueWith(block:
 {(task:AWSTask<AnyObject>) -> Any? in
    if let error = task.error as? NSError {
        if (error.domain == AWSLambdaInvokerErrorDomain) &&
 (AWSLambdaInvokerErrorType.functionError == AWSLambdaInvokerErrorType(rawValue:
 error.code) {
```

```
            print("Function error:
 \(error.userInfo[AWSLambdaInvokerFunctionErrorKey])")
        } else {
            print("Error: \(error)")
        }
        return nil
    }

    // Handle response in task.result
    if let JSONDictionary = task.result as? NSDictionary {
        print("Result: \(JSONDictionary)")
        print("resultKey: \(JSONDictionary["resultKey"])")
    }
    return nil
})
```

Objective C

```
AWSLambdaInvoker *lambdaInvoker = [AWSLambdaInvoker defaultLambdaInvoker];

[[lambdaInvoker invokeFunction:@"myFunction"
            JSONObject:@{@"key1" : @"value1",
                         @"key2" : @2,
                         @"key3" : [NSNull null],
                         @"key4" : @[@1, @"2"],
                         @"isError" : @NO}] continueWithBlock:^id(AWSTask *task) {
    if (task.error) {
        NSLog(@"Error: %@", task.error);
        if ([task.error.domain isEqualToString:AWSLambdaInvokerErrorDomain]
            && task.error.code == AWSLambdaInvokerErrorTypeFunctionError) {
            NSLog(@"Function error: %@",
 task.error.userInfo[AWSLambdaInvokerFunctionErrorKey]);
        }
    }
    if (task.result) {
        NSLog(@"Result: %@", task.result);
        NSDictionary *JSONObject = task.result;
        NSLog(@"result: %@", JSONObject[@"resultKey"]);
    }
    return nil;
}];
```

Client Context

Calls to AWS Lambda using this SDK provide your functions with data about the calling device and app using the *ClientContext* class.

You can access the client context in your lambda function as follows.

JavaScript

```
exports.handler = function(event, context) {
    console.log("installation_id = " + context.clientContext.client.installation_id);
    console.log("app_version_code = " + context.clientContext.client.app_version_code);
    console.log("app_version_name = " + context.clientContext.client.app_version_name);
    console.log("app_package_name = " + context.clientContext.client.app_package_name);
    console.log("app_title = " + context.clientContext.client.app_title);
    console.log("platform_version = " + context.clientContext.env.platform_version);
    console.log("platform = " + context.clientContext.env.platform);
    console.log("make = " + context.clientContext.env.make);
    console.log("model = " + context.clientContext.env.model);
    console.log("locale = " + context.clientContext.env.locale);
```

```
    context.succeed("Your platform is " + context.clientContext.env.platform;
}
```

ClientContext has the following fields:

client.installation_id

Auto-generated UUID that is created the first time the app is launched. This is stored in the keychain on the device. In case the keychain is wiped a new installation ID will be generated.

client.app_version_code

CFBundleShortVersionString

client.app_version_name

CFBundleVersion

client.app_package_name

CFBundleIdentifier

client.app_title

CFBundleDisplayName

env.platform_version

systemVersion

env.platform

systemName

env.make

Hardcoded as "apple"

env.model

Model of the device

env.locale

localeIdentifier from autoupdatingCurrentLocale

Identity Context

The *IdentityContext* class of the SDK passes Amazon Cognito credentials making the AWS identity of the end user available to your function. You can access the Identity ID as follows.

JavaScript

```
exports.handler = function(event, context) {
    console.log("clientID = " + context.identity);

    context.succeed("Your client ID is " + context.identity);
}
```

For more about Amazon Cognito in the AWS Mobile SDK for iOS, see :ref:`Amazon Cognito Setup for iOS <cognito-auth-aws-identity-for-ios>`.

How To Add Natural Language Understanding with Amazon Lex

Just Getting Started?	Use streamlined steps (p. 84) to install the SDK and integrate Amazon Lex.

Or, use the contents of this page if your app will integrate existing AWS services.

This section provides information on the steps for achieving specific tasks for integrating your Amazon Lex into your Android and iOS apps.

Topics

Android: Use Natural Language with Amazon Lex

Just Getting Started?	Use streamlined steps (p. 84) to install the SDK and integrate Amazon Lex.

Or, use the content on this page if your app integrates existing AWS services.

Overview

Amazon Lex is an AWS service for building voice and text conversational interfaces into applications. With Amazon Lex, the same natural language understanding engine that powers Amazon Alexa is now available to any developer, enabling you to build sophisticated, natural language chatbots into your new and existing applications.

The AWS Mobile SDK for Android provides an optimized client for interacting with Amazon Lex runtime APIs, which support both voice and text input and can return either voice or text. Amazon Lex has built-in integration with AWS Lambda to allow insertion of custom business logic into your Amazon Lex processing flow, including all of the extension to other services that Lambda makes possible.

For information on Amazon Lex concepts and service configuration, see How it Works in the *Lex Developer Guide*.

For information about Amazon Lex Region availability, see AWS Service Region Availability.

To get started using the Amazon Lex mobile client, integrate the SDK for Android into your app, set the appropriate permissions, and import the necessary libraries.

Setting Up

Include the SDK in Your Project

Follow the instructions at setup-legacy to include the JAR files for this service and set the appropriate permissions.

Set Permissions in Your Android Manifest

In your `AndroidManifest.xml` file, add the following permission:

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.RECORD_AUDIO" />
```

Declare Amazon Lex as a Gradle dependency

Make sure the following Gradle build dependency is declared in the `app/build.gradle` file.

```
implementation 'com.amazonaws:aws-android-sdk-lex:2.3.8@aar'
```

Set IAM Permissions for Amazon Lex

To use Amazon Lex in an application, create a role and attach policies as described in Step 1 of Getting Started in the *Lex Developer Guide*.

To learn more about IAM policies, see Using IAM.

Configure a Bot

Use the Amazon Lex console console to configure a bot that interacts with your mobile app features. To learn more, see Amazon Lex Developer Guide. For a quickstart, see Getting Started .

Amazon Lex also supports model building APIs, which allow creation of bots, intents, and slots at runtime. This SDK does not currently offer additional support for interacting with Amazon Lex model building APIs.

Implement Text and Voice Interaction with Amazon Lex

Get AWS User Credentials

Both text and voice API calls require validated AWS credentials. To establish Amazon Cognito as the credentials provider, include the following code in the function where you initialize your Amazon Lex interaction objects.

Android - Java

```
CognitoCredentialsProvider credentialsProvider =
   new CognitoCredentialsProvider(
       appContext.getResources().getString(R.string.identity_id_test),
       Regions.fromName(appContext.getResources().getString(R.string.aws_region))
   );
```

Android - Kotlin

```
val region = applicationContext.resources.getString(R.string.aws_region)
val credentialsProvider = CognitoCredentialsProvider
       applicationContext.resources.getString(R.string.identity_id_test),
       Regions.fromName(region))
```

Integrate Lex Interaction Client

Perform the following tasks to implement interaction with Lex in your Android app.

Initialize Your Lex Interaction Client

Instantiate an `InteractionClient`, providing the following parameters.

- The application context, credentials provider, and AWS Region
- `bot_name` - name of the bot as it appears in the Amazon Lex console
- `bot_alias` - the name associated with selected version of your bot
- `InteractionListener` - your app's receiver for text responses from Amazon Lex
- `AudioPlaybackListener` - your app's receiver for voice responses from Amazon Lex

Android - Java

```
// Create Lex interaction client.
lexInteractionClient = new InteractionClient(getApplicationContext(),
       credentialsProvider,
       Regions.US_EAST_1,
       <your_bot_name>,
       <your_bot_alias>);
lexInteractionClient.setAudioPlaybackListener(audioPlaybackListener);
lexInteractionClient.setInteractionListener(interactionListener);
```

Android - Kotlin

```
// Create Lex interaction client.
val lexInteractionClient = InteractionClient(applicationContext,
       credentialsProvider,
       Regions.US_EAST_1,
       <your_bot_name>,
       <your_bot_alias>)
lexInteractionClient.audioPlaybackListener = audioPlaybackListener
lexInteractionClient.interactionListener = interactionListener
```

Begin or Continue a Conversation

To begin a new conversation with Amazon Lex, we recommend that you clear any history of previous text interactions, and that you maintain a `inConversation` flag to make your app aware of when a conversation is in progress.

If `inConversation` is false when user input is ready to be sent as Amazon Lex input, then make a call using the `textInForTextOut`, `textInForAudioOut`, `audioInForTextOut`, or `audioInForAudioOut` method of an `InteractionClient` instance. These calls are in the form of:

Android - Java

```
lexInteractionClient.textInForTextOut(
   String text,
   Map<String, String> sessionAttributes)
```

Android - Kotlin

```
lexInteractionClient.textInForTextOut(
   text: String,
   sessionAttributes: Map<String,String>)
```

If `inConversation` is true, then the input should be passed to an instance of `LexServiceContinuation` using the `continueWithTextInForTextOut`, `continueWithTextInForAudioOut`, `continueWithAudioInForTextOut`, `continueWithAudioInForAudioOut` method. Continuation enables Amazon Lex to persist the state and metadata of an ongoing conversation across multiple interactions.

Interaction Response Events

`InteractionListener` captures a set of Amazon Lex response events that include:

- `onReadyForFulfillment(final Response response)`

  This response means that Lex has the information it needs to co fulfill the intent of the user and considers the transaction complete. Typically, your app would set your `inConversation` flag to false when this response arrives.
- `promptUserToRespond(final Response response, final LexServiceContinuation continuation)`

  This response means that Amazon Lex is providing the next piece of information needed in the conversation flow. Typically your app would pass the received continuation on to your Amazon Lex client.
- `onInteractionError(final Response response, final Exception e)`

  This response means that Amazon Lex is providing an identifier for the exception that has occured.

Microphone Events

`MicrophoneListener` captures events related to the microphone used for interaction with Amazon Lex that include:

- `startedRecording()`

  This event occurs when the user has started recording their voice input to Amazon Lex.
- `onRecordingEnd()`

  This event occurs when the user has finished recording their voice input to Amazon Lex.
- `onSoundLevelChanged(double soundLevel)`

  This event occurs when the volume level of audio being recorded changes.
- `onMicrophoneError(Exception e)`

  The event returns an exception when an error occurs while recording sound through the microphone.

Audio Playback Events

`AudioPlaybackListener` captures a set of events related to Amazon Lex voice responses that include:

- `onAudioPlaybackStarted()`

  This event occurs when playback of a Amazon Lex voice response starts.
- `onAudioPlayBackCompleted()`

  This event occurs when playback of a Amazon Lex voice response finishes.
- `onAudioPlaybackError(Exception e)`

  This event returns an exception when an error occurs duringplayback of an Amazon Lex voice response.

Add Voice Interactons

Perform the following tasks to implement voice interaction with Amazon Lex in your Android app.

`InteractiveVoiceView` simplifies the acts of receiving and playing voice responses from Lex by internally using the `InteractionClient` methods and both `MicrophoneListener` and

`AudioPlaybackListener` events described in the preceding sections. You can use those interfaces directly instead of instantiating `InteractiveVoiceView`.

Add a `voice-component` Layout Element to Your Activity

In the layout for your activity class that contains the voice interface for your app, include the following element.

```
<include
   android:id="@+id/voiceInterface"
   layout="@layout/voice_component"
   android:layout_width="200dp"
   android:layout_height="200dp"
    />
```

Initialize Your Voice Activity

In your activity class that contains the voice interface for your app, have the base class implement `InteractiveVoiceView.InteractiveVoiceListener`.

The following code shows initialization of `InteractiveVoiceView`.

Android - Java

```
private void init() {
    appContext = getApplicationContext();
    voiceView = (InteractiveVoiceView) findViewById(R.id.voiceInterface);
    voiceView.setInteractiveVoiceListener(this);
    CognitoCredentialsProvider credentialsProvider = new CognitoCredentialsProvider(
         <your_conginto_identity_pool_id>,
         Regions.fromName(<your_aws_region>)));
    voiceView.getViewAdapter().setCredentialProvider(credentialsProvider);
    voiceView.getViewAdapter().setInteractionConfig(
        new InteractionConfig(<your_bot_name>),  <your_bot_alias>));
    voiceView.getViewAdapter().setAwsRegion(<your_aws_region>));
}
```

Android - Kotlin

```
private fun init() {
   val voiceView = voiceInterface as InteractiveVoiceView
   val cp = CognitoCredentialsProvider(IDENTITY_POOL_ID, REGION)
   with (voiceView.viewAdapter) {
     credentialsProvider = cp
     setInteractionConfig(InteractionConfig(<your_bot_name>), <your_bot_alias>)
     setAwsRegion(REGION)
   }
}
```

iOS: Use Natural Language with Amazon Lex

Just Getting Started?	Use streamlined steps (p. 84) to install the SDK and integrate Amazon Lex.

Or, use the contents of this page if your app will integrate existing AWS services.

Overview

Amazon Lex is an AWS service for building voice and text conversational interfaces into applications. With Amazon Lex, the same natural language understanding engine that powers Amazon Alexa is now available to any developer, enabling you to build sophisticated, natural language chatbots into your new and existing applications.

The AWS Mobile SDK for iOS provides an optimized client for interacting with Amazon Lex runtime APIs, which support both voice and text input and can return either voice or text. Included are features like APIs to support detecting when a user finishes speaking and encoding incoming audio to the format the Amazon Lex service prefers.

Amazon Lex has built-in integration with AWS Lambda to allow insertion of custom business logic into your Amazon Lex processing flow, including all of the extension to other services that Lambda makes possible.

For information on Amazon Lex concepts and service configuration, see How it Works in the Amazon Lex Developer Guide.

To get started using the Amazon Lex mobile client for iOS, you'll need to integrate the SDK for iOS into your app, set the appropriate permissions, and import the necessary libraries.

Setting Up

Include the SDK in Your Project

Follow the instructions on the Set Up the SDK for iOS page to include the frameworks for this service.

Set IAM Permissions for Amazon Lex

To use Amazon Lex in an application, create a role and attach policies as described in Step 1 of Getting Started in the Amazon Lex Developer Guide.

To learn more about IAM policies, see Using IAM.

Configure a Bot

Use the Amazon Lex console console to configure a bot that interacts with your mobile app features. To learn more, see Amazon Lex Developer Guide. For a quickstart, see Getting Started .

Amazon Lex also supports model building APIs, which allow creation of bots, intents, and slots at runtime. This SDK does not currently offer additional support for interacting with Amazon Lex model building APIs.

Implement Text and Voice Interaction with Amazon Lex

Add Permissions and Get Credentials

Take the following steps to allow your app to access device resources and AWS services.

Add permission to use the microphone

To add permission to use the microphone to enable users to speak to Amazon Lex through your app, open your project's `Info.plist` file using **Right-click > Open As > Source Code**, and then add the following entry.

```
<plist version="1.0">
    . . .
    <dict>
        <key>NSMicrophoneUsageDescription</key>
        <string>For interaction with Amazon Lex</string>
    </dict>
     . . .
```

```
</plist>
```

Integrating the Interaction Client

Take the following steps to integrate the Amazon Lex interaction client with your app.

Initialize the *InteractionKit* for voice and text

Add the following code using the name and alias of your Lex bot to initialize an instance of *InteractionKit*.

iOS - Swift

```
let chatConfig = AWSLexInteractionKitConfig.defaultInteractionKitConfig(withBotName:
 BotName, botAlias: BotAlias)

// interaction kit for the voice button
AWSLexInteractionKit.register(with: configuration!, interactionKitConfiguration:
 chatConfig, forKey: "AWSLexVoiceButton")

chatConfig.autoPlayback = false

// interaction kit configuration for the client
AWSLexInteractionKit.register(with: configuration!, interactionKitConfiguration:
 chatConfig, forKey: "chatConfig")
```

Objective C

```
AWSLexInteractionKitConfig *chatConfig = [AWSLexInteractionKitConfig
 defaultInteractionKitConfigWithBotName:BotName botAlias:BotAlias];

chatConfig.autoPlayback = NO;

[AWSLexInteractionKit registerInteractionKitWithServiceConfiguration:configuration
 interactionKitConfiguration:chatConfig forKey:AWSLexChatConfigIdentifierKey];
```

Implement *InteractionKit* delegate methods

Declare and implement the following methods in the class where you intend to use your *InteractionKit*:

- `interactionKit` is called to begin a conversation. When passed `interactionKit`, `switchModeInput`, and `completionSource`, the function should set the mode of interaction (audio or text input and output) and pass the `SwitchModeResponse` to the `completionSource`. On error, the *interactionKit:onError* method is called.

  iOS - Swift

  ```
  public func interactionKit(_ interactionKit: AWSLexInteractionKit, switchModeInput:
    AWSLexSwitchModeInput, completionSource:
   AWSTaskCompletionSource<AWSLexSwitchModeResponse>?)

  public func interactionKit(_ interactionKit: AWSLexInteractionKit, onError error:
   Error)
  ```

  Objective C

  ```
  - (void)interactionKit:(AWSLexInteractionKit *)interactionKit
      switchModeInput:(AWSLexSwitchModeInput *)switchModeInput
    completionSource:(AWSTaskCompletionSource<AWSLexSwitchModeResponse *>
   *)completionSource
  ```

```
- (void)interactionKit:(AWSLexInteractionKit *)interactionKit
  onError:(NSError *)error`
```

- `interactionKitContinue` is called to continue an ongoing conversation with its transaction state and metadata maintained.

  iOS - Swift

  ```
  func interactionKitContinue(withText interactionKit: AWSLexInteractionKit,
   completionSource: AWSTaskCompletionSource<NSString>){
      textModeSwitchingCompletion = completionSource
  }
  ```

  Objective C

  ```
  - (void)interactionKitContinueWithText:(AWSLexInteractionKit *)interactionKit
      completionSource:(AWSTaskCompletionSource<NSString *> *)completionSource{
   textModeSwitchingCompletion = completionSource;
  }
  ```

  Alternatively, you can explicitly set *SwitchModeResponse* to a selected mode.

  iOS - Swift

  ```
  let switchModeResponse = AWSLexSwitchModeResponse()
  switchModeResponse.interactionMode = AWSLexInteractionMode.text
  switchModeResponse.sessionAttributes = switchModeInput.sessionAttributes
  completionSource?.setResult(switchModeResponse)
  ```

  Objective C

  ```
  AWSLexSwitchModeResponse *switchModeResponse = [AWSLexSwitchModeResponse new];
  [switchModeResponse setInteractionMode:AWSLexInteractionModeText];
  [switchModeResponse setSessionAttributes:switchModeInput.sessionAttributes];
  [completionSource setResult:switchModeResponse];
  ```

Begin or Continue a Conversation

When you call `InteractionKit` to provide input for a conversation, check if the conversation is already in progress by examining the state of `AWSTaskCompletionSource`. The following example illustrates the case where `textModeSwitchingCompletion` is an `AWSTaskCompletionSource` instance and the desired result is that a new conversation will be in the `texttInTextOut` mode.

iOS - Swift

```
if let textModeSwitchingCompletion = textModeSwitchingCompletion {
        textModeSwitchingCompletion.setResult(text)
        self.textModeSwitchingCompletion = nil
    }
    else {
        self.interactionKit?.textInTextOut(text)
    }
```

Objective C

```
if(textModeSwitchingCompletion){
    [textModeSwitchingCompletion setResult:text];
```

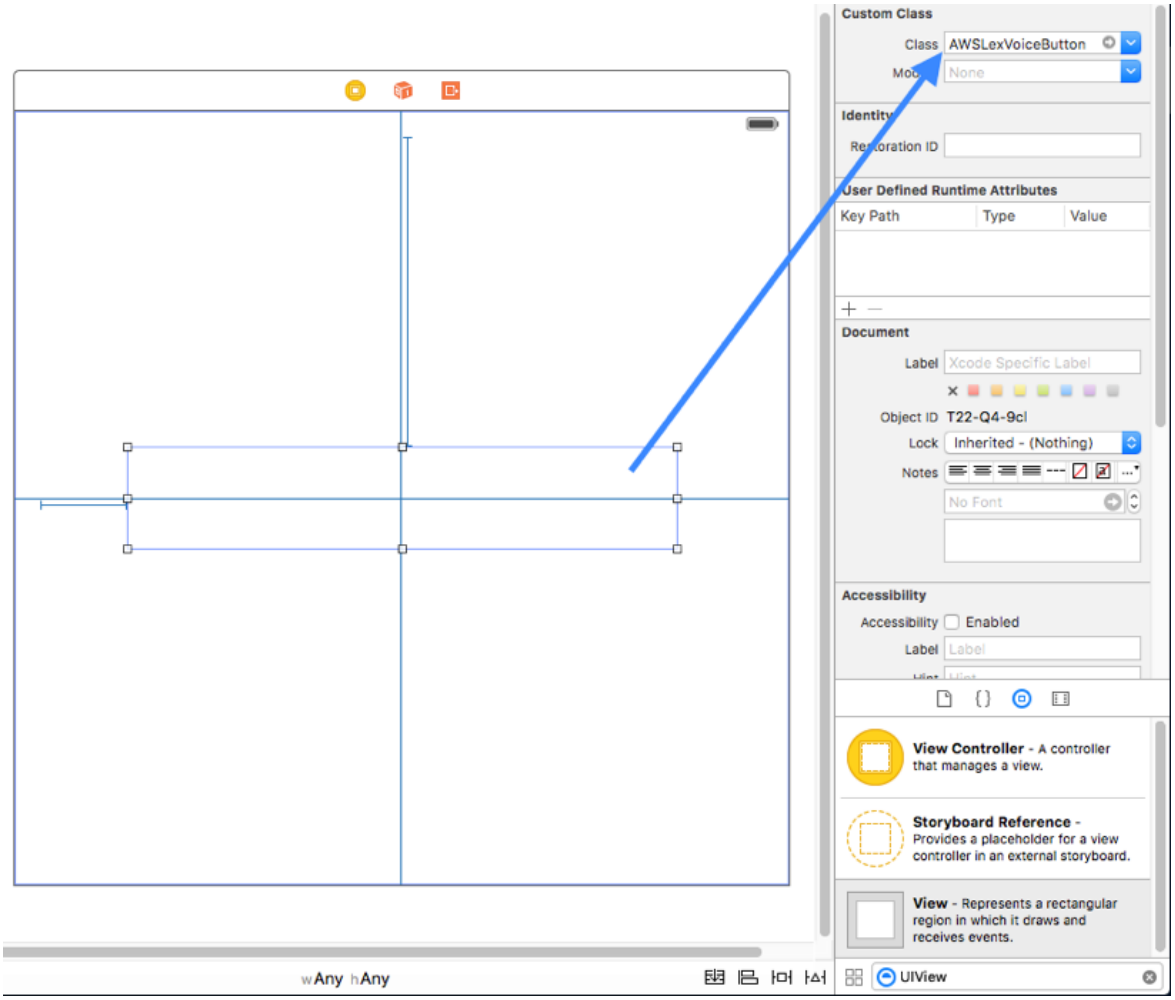
```
    textModeSwitchingCompletion = nil;
  }else{
    [self.interactionKit textInTextOut:text];
}
```

Integrating Voice Conversation

Perform the following tasks to implement voice interaction with Amazon Lex in your iOS app.

Add a voice button and bind it to the Lex SDK UI component

Add a voice UIView into your storyboard scene or xib file, add a voice button (the UI element that enables users to speak to Amazon Lex). Map the voice button to the SDK button component by setting the *class* for the voice UIView to *AWSLexVoiceButton* as illustrated in the following image.



Convert Text to Speech with Amazon Polly

What is Amazon Polly?

Amazon Polly is a service that turns text into lifelike speech, making it easy to develop mobile applications that use high-quality speech to increase engagement and accessibility. With Amazon Polly you can quickly build speech-enabled apps that work in multiple geographies.

Using the following resources, you can integrate Amazon Polly with your iOS app to add text to speech transformation. No deep knowledge of either AWS services or natural language computing is needed.

For information on Amazon Polly concepts and service configuration, see How it Works in the Amazon Polly Developer Guide.

ANDROID

- For sample Android integration code, see the Android Amazon Polly Example.
- For end to end sample apps using Amazon Polly see the AWS SDK for Android samples.

iOS

- For sample iOS integration code, see the iOS Amazon Polly Example.
- For more end to end sample apps using Amazon Polly see the AWS SDK for iOS samples.

How To Stream Data with Amazon Kinesis

This section provides information on the steps for achieving specific tasks for integrating your Amazon Kinesis into your Android and iOS apps.

Topics

Android: Process Data Streams with Kinesis

Overview

Amazon Kinesis is a fully managed service for real-time processing of streaming data at massive scale. Kinesis can collect and process hundreds of terabytes of data per hour from hundreds of thousands of sources, so you can write applications that process information in real-time. With Kinesis applications, you can build real-time dashboards, capture exceptions and generate alerts, drive recommendations, and make other real-time business or operational decisions. You can also easily send data to other services such as Amazon Simple Storage Service, Amazon DynamoDB, and Amazon Redshift.

The AWS Mobile SDK for Android provides simple, high-level clients designed to help you interface with Kinesis. The Kinesis clients let you store streaming data on disk and then send them all at once. This is useful because many mobile applications that use Kinesis will create multiple data requests per second. Sending one data request for each action could adversely impact battery life. Moreover, the requests could be lost if the device goes offline. Thus, using the high-level Kinesis client for batching can preserve both battery life and data.

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

To get started using the Amazon Kinesis mobile client, you'll need to integrate the SDK for Android into your app, set the appropriate permissions, and import the necessary libraries.

What is Kinesis Data Firehose?

Amazon Kinesis Data Firehose is a fully managed service for delivering real-time streaming data to destinations such as Amazon S3 and Amazon Redshift. With Kinesis Data Firehose, you do not need to write any applications or manage any resources. You configure your data producers to send data to Firehose and it automatically delivers the data to the destination that you specified.

KinesisFirehoseRecorder is the high level client for Kinesis Data Firehose. Its usage is very similar to that of **KinesisRecorder**.

For more information about Kinesis Data Firehose, see Amazon Kinesis Firehose.

You can also learn more about how the Kinesis services work together on the following page: Amazon Kinesis services.

Getting Started

Create an Identity Pool

To use AWS services in your mobile application, you must obtain AWS Credentials using Amazon Cognito Identity as your credential provider. Using a credentials provider allows you to access AWS services without having to embed your private credentials in your application. This also allows you to set permissions to control which AWS services your users have access to.

The identities of your application's users are stored and managed by an identity pool, which is a store of user identity data specific to your account. Every identity pool has roles that specify which AWS resources your users can access. Typically, a developer will use one identity pool per application. For more information on identity pools, see the Cognito Developer Guide.

To create an identity pool for your application:

1. Log in to the Cognito Console and click **Create new identity pool**.
2. Enter a name for your Identity Pool and check the checkbox to enable access to unauthenticated identities. Click **Create Pool** to create your identity pool.
3. Click **Allow** to create the roles associated with your identity pool.

The next page displays code that creates a credentials provider so you can easily integrate Amazon Cognito Identity in your Android application.

For more information on Cognito Identity, see cognito-auth.

Set IAM Permissions (Amazon Kinesis)

To use Amazon Kinesis in an application, you must set the correct permissions. The following IAM policy allows the user to submit records to a Kinesis stream identified by ARN:

```
{
    "Statement": [{
        "Effect": "Allow",
        "Action": "kinesis:PutRecords",
        "Resource": "arn:aws:kinesis:us-west-2:111122223333:stream/mystream"
    }]
}
```

This policy should be applied to roles assigned to the Cognito identity pool, but you will need to replace the `Resource` value with the correct ARN for your Kinesis stream. You can apply policies at the IAM console.

Set IAM Permissions (Amazon Kinesis Firehose)

Amazon Kinesis Firehose needs slightly different permission. The following IAM policy allows the user to submit records to an Amazon Kinesis Firehose stream identified by the Amazon Resource Name (ARN):

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

```
        "Effect": "Allow",
        "Action": "firehose:PutRecordBatch",
        "Resource": "arn:aws:firehose:us-west-2:111122223333:deliverystream/mystream"
    }]
}
```

For more information about ARN formatting and example policies, see Amazon Resource Names for Amazon Kinesis.

To learn more about Kinesis-specific policies, see Controlling Access to Amazon Kinesis Resources with IAM.

To learn more about IAM policies, see Using IAM.

Include the SDK in Your Project

Follow the instructions on the Set Up the SDK for Android page to include the proper JAR files for this service and set the appropriate permissions.

Set Permissions in Your Android Manifest

In your `AndroidManifest.xml` file, add the following permission:

```
<uses-permission android:name="android.permission.INTERNET" />
```

Add Import Statements

Add the following imports to the main activity of your app.

```
import com.amazonaws.mobileconnectors.kinesis.kinesisrecorder.*;
import com.amazonaws.auth.CognitoCachingCredentialsProvider;
import com.amazonaws.regions.Regions;
```

Instantiate a Kinesis recorder

Once you've imported the necessary libraries and have your credentials object, you can instantiate `KinesisRecorder`. `KinesisRecorder` is a high-level client meant for storing PutRecord requests on an Android device. Storing requests on the device lets you retain data when the device is offline, and it can also increase performance and battery efficiency since the network doesn't need to be awakened as frequently.

> **Note**
> `KinesisRecorder` uses synchronous calls, so you shouldn't call `KinesisRecorder` methods on the main thread.

When you create the `KinesisRecorder` client, you'll pass in a directory and an AWS region. The directory should be empty the first time you instantiate `KinesisRecorder`; it should be private to your application; and, to prevent collision, it should be used only by `KinesisRecorder`. The following snippet creates a directory and instantiates the `KinesisRecorder` client, passing in a Cognito credentials object (`cognitoProvider`), a region enum, and the directory.

Android - Java

```
String kinesisDirectory = "YOUR_UNIQUE_DIRECTORY";
KinesisRecorder recorder = new KinesisRecorder(
  myActivity.getDir(kinesisDirectory, 0),
  Regions.US_WEST_2,
  credentialsProvider
);
```

You'll use `KinesisRecorder` to save records and then send them in a batch.

```
recorder.saveRecord("MyData".getBytes(),"MyStreamName");
recorder.submitAllRecords();
```

> **Note**
> For the `saveRecord()` request above to work, you would have to have created a stream named **MyStreamName**. You can create new streams in the Amazon Kinesis console.

If `submitAllRecords()` is called while the app is online, requests will be sent and removed from the disk. If `submitAllRecords()` is called while the app is offline, requests will be kept on disk until `submitAllRecords()` is called while online. This applies even if you lose your internet connection midway through a submit. So if you save ten requests, call `submitAllRecords()`, send five, and then lose the Internet connection, you have five requests left on disk. These remaining five will be sent the next time `submitAllRecords()` is invoked online.

To see how much space the `KinesisRecorder` client is allowed to use, you can call `getDiskByteLimit()`.

```
Long byteLimit = recorder.getDiskByteLimit();
// Do something with byteLimit
```

Alternatively, you can retrieve the same information by getting the `KinesisRecorderConfig` object for the recorder and calling `getMaxStorageSize()`:

```
KinesisRecorderConfig kinesisRecorderConfig = recorder.getKinesisRecorderConfig();
Long maxStorageSize = kinesisRecorderConfig.getMaxStorageSize();
// Do something with maxStorageSize
```

Android - Kotlin

```
val recorder = KinesisRecorder(
           myActivity.getDir("YOUR_UNIQUE_DIRECTORY", 0),
           Regions.US_WEST_2,
           credentialsProvider)
```

You'll use `KinesisRecorder` to save records and then send them in a batch.

```
recorder.saveRecord("MyData".getBytes(), "MyStreamName")
recorder.submitAllRecords()
```

> **Note**
> For the `saveRecord()` request above to work, you would have to have created a stream named **MyStreamName**. You can create new streams in the Amazon Kinesis console.

If `submitAllRecords()` is called while the app is online, requests will be sent and removed from the disk. If `submitAllRecords()` is called while the app is offline, requests will be kept on disk until `submitAllRecords()` is called while online. This applies even if you lose your internet connection midway through a submit. So if you save ten requests, call `submitAllRecords()`, send five, and then lose the Internet connection, you have five requests left on disk. These remaining five will be sent the next time `submitAllRecords()` is invoked online.

To see how much space the `KinesisRecorder` client is allowed to use, you can call `getDiskByteLimit()`.

```
val byteLimit = recorder.diskByteLimit
```

```
// Do something with byteLimit
```

Alternatively, you can retrieve the same information by getting the `KinesisRecorderConfig` object for the recorder and calling `getMaxStorageSize()`:

```
val maxStorageSize = recorder.kinesisRecorderConfig.maxStorageSize
// Do something with maxStorageSize
```

Storage limits

If you exceed the storage limit for `KinesisRecorder`, requests will not be saved or sent. `KinesisRecorderConfig` has a default `maxStorageSize` of 8 MiB. You can configure the maximum allowed storage via the `withMaxStorageSize()` method of `KinesisRecorderConfig`.

To check the number of bytes currently stored in the directory passed in to the `KinesisRecoder` constructor, call `getDiskBytesUsed()`:

Android - Java

```
Long bytesUsed = recorder.getDiskBytesUsed();
// Do something with bytesUsed
```

Android - Kotlin

```
val bytesUsed = recorder.diskBytesUsed
// Do something with bytesUsed
```

To learn more about working with Amazon Kinesis, see Amazon Kinesis Developer Resources. To learn more about the Kinesis classes, see the API Reference for the Android SDK.

Use KinesisFirehoseRecorder

To use `KinesisFirehoseRecorder`, you need to pass the object in a directory where streaming data is saved. We recommend you use an app private directory because the data is not encrypted.

Android - Java

```
// Gets a working directory for the recorder
File directory = context.getCachedDir();
// Sets Firehose region
Regions region = Regions.US_WEST_2;
// Initialize a credentials provider to access Amazon Kinesis Firehose
AWSCredentialsProvider provider = new CognitoCachingCredentialsProvider(
     context, "identityPoolId",
     Regions.US_EAST_1); // region of your Amazon Cognito identity pool
KinesisFirehoseRecorder firehoseRecorder = new KinesisFirehoseRecorder(
     directory, region, provider);

// Start to save data, either a String or a byte array
firehoseRecorder.saveRecord("Hello world!\n");
firehoseRecorder.saveRecord("Streaming data to Amazon S3 via Amazon Kinesis Firehose is
 easy.\n");

// Send previously saved data to Amazon Kinesis Firehose
// Note: submitAllRecords() makes network calls, so wrap it in an AsyncTask.
new AsyncTask<Void, Void, Void>() {
    @Override
    protected Void doInBackground(Void... v) {
```

```
        try {
            firehoseRecorder.submitAllRecords();
        } catch (AmazonClientException ace) {
            // handle error
        }
    }
}.execute();
```

Android - Kotlin

```
val firehose = KinesisFirehoseRecorder(
   context.getCachedDir(),     // Working directory for recorder
   Regions.US_WEST_2,          // Region that Kinesis is provisioned in
   credentialsProvider)        // AWS Credentials provider

// Start to save data, either a String or a byte array
firehose.saveRecord("Hello world!\n");
firehose.saveRecord("Streaming data to Amazon S3 via Amazon Kinesis Firehose is easy.
\n");

// Send previously saved data to Amazon Kinesis Firehose
// Note: submitAllRecords() makes network calls.
thread(start = true) {
   try {
       firehose.submitAllRecords()
   } catch (ex: AmazonClientException) {
       Log.e(TAG, "Error submitting records")
   }
}
```

To learn more about working with Amazon Kinesis Firehose, see Amazon Kinesis Firehose.

To learn more about the Kinesis Firehose classes, see the API Reference for the Android SDK.

ios: Process Data Streams with Amazon Kinesis

Overview

Amazon Kinesis is a fully managed service for real-time processing of streaming data at massive scale.

The SDK for iOS provides two high-level client classes, **`AWSKinesisRecorder`** and **`AWSFirehoseRecorder`** , designed to help you interface with Amazon Kinesis and Amazon Kinesis Firehose.

The Amazon Kinesis **`AWSKinesisRecorder`** client lets you store PutRecord requests on disk and then send them all at once. This is useful because many mobile applications that use Amazon Kinesis will create multiple **`PutRecord`** requests per second. Sending an individual request for each **`PutRecord`** action could adversely impact battery life. Moreover, the requests could be lost if the device goes offline. Thus, using the high-level Amazon Kinesis client for batching can preserve both battery life and data.

The Amazon Kinesis Firehose **`AWSFirehoseRecorder`** client lets you store PutRecords requests on disk and then send them using **`PutRecordBatch`** .

For information about Amazon Kinesis Region availability, see AWS Service Region Availability.

What is Amazon Kinesis?

Amazon Kinesis is a fully managed service for real-time processing of streaming data at massive scale. Amazon Kinesis can collect and process hundreds of terabytes of data per hour from hundreds of thousands of sources, so you can write applications that process information in real-time. With Amazon

Kinesis applications, you can build real-time dashboards, capture exceptions and generate alerts, drive recommendations, and make other real-time business or operational decisions. You can also easily send data to other services such as Amazon Simple Storage Service, Amazon DynamoDB, and Amazon Redshift.

What is Amazon Kinesis Firehose?

Amazon Kinesis Firehose is a fully managed service for delivering real-time streaming data to destinations such as Amazon Simple Storage Service (Amazon S3) and Amazon Redshift. With Firehose, you do not need to write any applications or manage any resources. You configure your data producers to send data to Firehose and it automatically delivers the data to the destination that you specified.

For more information about Amazon Kinesis Firehose, see Amazon Kinesis Firehose.

You can also learn more about how the Amazon Kinesis services work together on the following page: Amazon Kinesis services.

Integrating Amazon Kinesis and Amazon Kinesis Firehose

To use the Amazon Kinesis mobile client, you'll need to integrate the SDK for iOS into your app and import the necessary libraries. To do so, follow these steps:

If you haven't already done so, download the SDK for iOS, unzip it, and include it in your application as described at setup-aws-sdk-for-ios. The instructions direct you to import the headers for the services you'll be using. For this example, you need the following import.

Swift

```
import AWSKinesis
```

Objective-C

```
#import <AWSKinesis/AWSKinesis.h>
```

You can use Amazon Cognito to provide temporary AWS credentials to your application.

These credentials let the app access your AWS resources. To create a credentials provider, follow the instructions at Cognito Identity Developer Guide.

To use Amazon Kinesis in an application, you must set the correct permissions. The following IAM policy allows the user to submit records to a specific Amazon Kinesis stream, which is identified by ARN.

```
{
    "Statement": [{
        "Effect": "Allow",
        "Action": "kinesis:PutRecords",
        "Resource": "arn:aws:kinesis:us-west-2:111122223333:stream/mystream"
    }]
}
```

The following IAM policy allows the user to submit records to a specific Amazon Kinesis Firehose stream.

```
{
    "Statement": [{
        "Effect": "Allow",
        "Action": "firehose:PutRecordBatch",
        "Resource": "arn:aws:firehose:us-west-2:111122223333:deliverystream/mystream"
    }]
```

```
}
```

This policy should be applied to roles assigned to the Amazon Cognito identity pool, but you will need to replace the **`Resource`** value with the correct ARN for your Amazon Kinesis or Amazon Kinesis Firehose stream. You can apply policies at the IAM console. To learn more about IAM policies, see Using IAM.

To learn more about Amazon Kinesis-specific policies, see Controlling Access to Amazon Kinesis Resources with IAM.

To learn more about Amazon Kinesis Firehose policies, see Controlling Access with Amazon Kinesis Firehose.

Once you have credentials, you can use **`AWSKinesisRecorder`** with Amazon Kinesis. The following snippet returns a shared instance of the Amazon Kinesis service client:

Swift

```
let kinesisRecorder = AWSKinesisRecorder.default()
```

Objective-C

```
AWSKinesisRecorder *kinesisRecorder = [AWSKinesisRecorder defaultKinesisRecorder];
```

You can use **`AWSFirehoseRecorder`** with Amazon Kinesis Firehose. The following snippet returns a shared instance of the Amazon Kinesis Firehose service client:

Swift

```
let firehoseRecorder = AWSFirehoseRecorder.default()
```

Objective-C

```
AWSFirehoseRecorder *firehoseRecorder = [AWSFirehoseRecorder defaultFirehoseRecorder];
```

You can configure **`AWSKinesisRecorder`** or **`AWSFirehoseRecorder`** through their properties:

Swift

```
kinesisRecorder.diskAgeLimit = TimeInterval(30 * 24 * 60 * 60); // 30 days
kinesisRecorder.diskByteLimit = UInt(10 * 1024 * 1024); // 10MB
kinesisRecorder.notificationByteThreshold = UInt(5 * 1024 * 1024); // 5MB
```

Objective-C

```
kinesisRecorder.diskAgeLimit = 30 * 24 * 60 * 60; // 30 days
kinesisRecorder.diskByteLimit = 10 * 1024 * 1024; // 10MB
kinesisRecorder.notificationByteThreshold = 5 * 1024 * 1024; // 5MB
```

The **`diskAgeLimit`** property sets the expiration for cached requests. When a request exceeds the limit, it's discarded. The default is no age limit. The **`diskByteLimit`** property holds the limit of the disk cache size in bytes. If the storage limit is exceeded, older requests are discarded. The default value is 5 MB. Setting the value to 0 means that there's no practical limit. The **`notficationByteThreshold`** property sets the point beyond which Kinesis issues a notification that the byte threshold has been reached. The default value is 0, meaning that by default Amazon Kinesis doesn't post the notification.

To see how much local storage is being used for Amazon Kinesis **`PutRecord`** requests, check the **`diskBytesUsed`** property.

With **`AWSKinesisRecorder`** created and configured, you can use **`saveRecord:streamName:`** to save records to local storage.

Swift

```
let yourData = "Test_data".data(using: .utf8)
kinesisRecorder.saveRecord(yourData, streamName: "YourStreamName")
```

Objective-C

```
NSData *yourData = [@"Test_data" dataUsingEncoding:NSUTF8StringEncoding];
[kinesisRecorder saveRecord:yourData streamName:@"YourStreamName"]
```

In the preceding example, we create an NSData object and save it locally. **`YourStreamName`** should be a string corresponding to the name of your Kinesis stream. You can create new streams in the Amazon Kinesis console.

Here is a similar snippet for Amazon Kinesis Firehose:

Swift

```
let yourData = "Test_data".data(using: .utf8)
firehoseRecorder.saveRecord(yourData, streamName: "YourStreamName")
```

Objective-C

```
NSData *yourData = [@"Test_data" dataUsingEncoding:NSUTF8StringEncoding];
[firehoseRecorder saveRecord:yourData streamName:@"YourStreamName"]
```

To submit all the records stored on the device, call **`submitAllRecords`** .

Swift

```
kinesisRecorder.submitAllRecords()

firehoseRecorder.submitAllRecords()
```

Objective-C

```
[kinesisRecorder submitAllRecords];

[firehoseRecorder submitAllRecords];
```

`submitAllRecords` sends all locally saved requests to the Amazon Kinesis service. Requests that are successfully sent will be deleted from the device. Requests that fail because the device is offline will be kept and submitted later. Invalid requests are deleted.

Both **`saveRecord`** and **`submitAllRecords`** are asynchronous operations, so you should ensure that **`saveRecord`** is complete before you invoke **`submitAllRecords`** . The following code sample shows the methods used correctly together.

Swift

```
// Create an array to store a batch of objects.
var tasks = Array<AWSTask<AnyObject>>()
for i in 0...100 {
    tasks.append(kinesisRecorder!.saveRecord(String(format: "TestString-%02d",
 i).data(using: .utf8), streamName: "YourStreamName")!)
}

AWSTask(forCompletionOfAllTasks: tasks).continueOnSuccessWith(block:
 { (task:AWSTask<AnyObject>) -> AWSTask<AnyObject>? in
    return kinesisRecorder?.submitAllRecords()
}).continueWith(block: { (task:AWSTask<AnyObject>) -> Any? in
    if let error = task.error as? NSError {
        print("Error: \(error)")
    }
    return nil
})
```

Objective-C

```
// Create an array to store a batch of objects.
NSMutableArray *tasks = [NSMutableArray new];
for (int32_t i = 0; i < 100; i++) {
    [tasks addObject:[kinesisRecorder saveRecord:[[NSString
 stringWithFormat:@"TestString-%02d", i] dataUsingEncoding:NSUTF8StringEncoding]
                                  streamName:@"YourStreamName"]];
}
[[[AWSTask taskForCompletionOfAllTasks:tasks] continueWithSuccessBlock:^id(AWSTask
 *task) {
    return [kinesisRecorder submitAllRecords];
}] continueWithBlock:^id(AWSTask *task) {
    if (task.error) {
        NSLog(@"Error: [%@]", task.error);
    }
    return nil;
}];
```

To learn more about working with Amazon Kinesis, see the Amazon Kinesis Developer Resources.

To learn more about the Amazon Kinesis classes, see the class reference for AWSKinesisRecorder.

For information about AWS service region availability, see AWS Service Region Availability.

How To: Sync Data with Amazon Cognito Sync

New User?	Use AWS AppSync instead. AppSync is a new service for synchronizing application data across devices. Like Cognito Sync, AppSync enables synchronization of a user's own data, such as game state or app preferences. AppSync extends these capabilities by allowing multiple users to synchronize and collaborate in real-time on shared data, such as a virtual meeting space or chatroom. Start building with AWS AppSync now.

Existing Cognito Sync users can find documentation here:

Topics

Android: Sync Data with Amazon Cognito Sync

New User?	Use AWS AppSync instead. AppSync is a new service for synchronizing application data across devices. Like Cognito Sync, AppSync enables synchronization of a user's own data, such as game state or app preferences. AppSync extends these capabilities by allowing multiple users to synchronize and collaborate in real-time on shared data, such as a virtual meeting space or chatroom. Start building an Android app with AWS AppSync now.

Overview

Amazon Cognito Sync is an AWS service and client library that enables cross-device syncing of application-related user data. You can use the Amazon Cognito Sync API to synchronize user profile data across devices and across login providers—Amazon, Facebook, Twitter/Digits, Google, and your own custom identity provider.

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

Set Up the SDK

You must complete all of the instructions on the Android: Setup Options for the SDK (p. 133) page before beginning this tutorial.

Initialize the CognitoSyncManager

Pass your initialized Amazon Cognito credentials provider to the `CognitoSyncManager` constructor:

```
CognitoSyncManager client = new CognitoSyncManager(
    getApplicationContext(),
    Regions.YOUR_REGION,
    credentialsProvider);
```

For more information about Cognito Identity, see cognito-auth-legacy.

Syncing User Data

To sync unauthenticated user data:

1. Create a dataset and add user data.
2. Synchronize the dataset with the cloud.

Create a Dataset and Add User Data

Synchronize Dataset with the Cloud

To synchronize a dataset, call its synchronize method:

```
dataset.synchronize();
```

All data written to datasets will be stored locally until the dataset is synced. The code in this section assumes you are using an unauthenticated Cognito identity, so when the user data is synced with the cloud it will be stored per device. The device has a device ID associated with it. When the user data is synced to the cloud, it will be associated with that device ID.

To sync user data across devices (using an authenticated identity), see Amazon Cognito Sync.

iOS: Sync Data with Amazon Cognito Sync

New User?	Use AWS AppSync instead. AppSync is a new service for synchronizing application data across devices. Like Cognito Sync, AppSync enables synchronization of a user's own data, such as game state or app preferences. AppSync extends these capabilities by allowing multiple users to synchronize and collaborate in real-time on shared data, such as a virtual meeting space or chatroom. Start building an iOS app with AWS AppSync now.

Authenticate Users with Amazon Cognito Identity

Amazon Cognito Identity provides secure access to AWS services. Identities are managed by an identity pool. Roles specify resources an identity can access and are associated with an identity pool. To create an identity pool for your application:

1. Log into the Amazon Cognito console and click the **New Identity Pool** button
2. Give your Identity Pool a unique name and enable access to unauthenticated identities
3. Click the **Create Pool** button and then the **Update Roles** to create your identity pool and associated roles

For more information on Amazon Cognito Identity, see Amazon Cognito Setup for iOS.

> **Note**
> The auto-generated Roles include the permissions needed to access Amazon Cognito Sync, so no further configuration is required.

The next page displays code that creates a credential provider that provides a Amazon Cognito Identity for your app to use. Copy the code from Steps 1 & 2 into your AppDelegate.m file as shown below:

Add the following import statements:

Swift

```
import AWSCore
```

```
import AWSCognito
```

Objective-C

```
#import <AWSCore/AWSCore.h>
#import <AWSCognito/AWSCognito.h>
```

If you have an existing AWS credential provider, add the following code to *application:didFinishLaunchingWithOptions* method:

Swift

```
let credentialProvider = AWSCognitoCredentialsProvider(regionType: .USEast1,
 identityPoolId: "YourIdentityPoolId")
let configuration = AWSServiceConfiguration(region: .USEast1, credentialsProvider:
 credentialProvider)
AWSServiceManager.default().defaultServiceConfiguration = configuration
```

Objective-C

```
AWSCognitoCredentialsProvider *credentialsProvider = [[AWSCognitoCredentialsProvider
 alloc] initWithRegionType:AWSRegionUSEast1
identityPoolId:@"<your-identity-pool-arn>"];

AWSServiceConfiguration *configuration = [[AWSServiceConfiguration alloc]
 initWithRegion:AWSRegionUSEast1 credentialsProvider:credentialsProvider];

AWSServiceManager.defaultServiceManager.defaultServiceConfiguration = configuration;
```

For more information on Amazon Cognito Identity, see Amazon Cognito Setup for iOS

Syncing User Data

To sync unauthenticated user data:

1. Create a dataset and add user data.
2. Synchronize the dataset with the cloud.

Create a Dataset and Add User Data

Create an instance of `AWSCognitoDataset`. User data is added in the form of key/value pairs. Dataset objects are created with the `AWSCognito` class which functions as a Amazon Cognito client object. Use the defaultCognito method to get a reference to the default singleton instance of AWSCognito. The openOrCreateDataset method is used to create a new dataset or open an existing instance of a dataset stored locally on the device:

Swift

```
let dataset = AWSCognito.default().openOrCreateDataset("user_data")
```

Objective-C

```
AWSCognitoDataset *dataset = [[AWSCognito defaultCognito]
 openOrCreateDataset:datasetName];:@"user_data"];
```

User data is added to an AWSCognitoDataset instance using the setString:forKey or setValue:forKey methods. The following code snippet shows how to add some user data to a dataset:

Swift

```
dataset?.setString("John Doe", forKey:"Username")
dataset?.setString("10000", forKey:"HighScore")
```

Objective-C

```
[dataset setString:@"John Doe" forKey:@"Username"];
[dataset setString:@"10000" forKey:@"HighScore"];
```

Synchronize Dataset with the Cloud

To sync the dataset with the cloud, call the synchronize method on the dataset object:

Swift

```
_ = dataset?.synchronize()
```

Objective-C

```
[dataset synchronize];
```

All data written to datasets will be stored locally until the dataset is synced. The code in this section assumes you are using an unauthenticated Amazon Cognito identity, so when the user data is synced with the cloud it will be stored per device. The device has a device ID associated with it, when the user data is synced to the cloud, it will be associated with that device ID.

To sync user data across devices (based on an authenticated Cognito Identity) see Amazon Cognito Sync Developer Guide.

How To Add Machine Learning with Amazon Machine Learning

This section provides information on the steps for achieving specific tasks for integrating your Amazon Machine Learning into your Android and iOS apps.

Topics

Android: Amazon Machine Learning

Amazon Machine Learning (ML) is a service that makes it easy for developers of all skill levels to use machine learning technology. The SDK for Android provides a simple, high-level client designed to help you interface with Amazon Machine Learning service. The client enables you to call Amazon ML's real-time API to retrieve predictions from your models and enables you to build mobile applications that request and take actions on predictions. The client also enables you to retrieve the real-time prediction endpoint URLs for your ML models.

Setup

Prerequisites

You must complete all of the instructions on the Set Up the SDK for Android page before beginning this tutorial.

Granting Access to Amazon Machine Learning Resources

To use Amazon Machine Learning in an application, you must set the proper permissions. The following IAM policy allows the user to perform the actions shown in this tutorial on two actions identified by ARN

```
{
   "Statement": [{
   "Effect": "Allow",
   "Action": [
       "machinelearning:GetMLModel",
       "machinelearning:Predict"
   ],
   "Resource": "arn:aws:machinelearning:use-east-1:11122233444:mlmodel/example-model-id"
 }]
}
```

This policy should be applied to roles assigned to the Amazon Cognito identity pool, but you will need to replace the Resource value with the correct account ID and ML Model ID. You can apply policies at the IAM console. To learn more about IAM policies, see Introduction to IAM.

Add Import Statements

Add the following imports to the main activity of your app:

```
import com.amazonaws.services.machinelearning.*;
```

Initialize AmazonMachineLearningClient

Pass your initialized Amazon Cognito credentials provider to the `AmazonMachineLearningClient` constructor:

Android - Java

```
AmazonMachineLearningClient client = new
 AmazonMachineLearningClient(credentialsProvider);
```

Android - Kotlin

```
val client = AmazonMachineLearningClient(credentialsProvider)
```

Create an Amazon Machine Learning Client

Making a Predict Request

Prior to calling Predict, make sure you have not only a completed ML Model ID but also a created real-time endpoint for that ML Model ID. This cannot be done through the mobile SDK; you will have to use the Machine Learning Console or an alternate SDK. To validate that this ML can be used for real-time Predictions:

Android - Java

```
// Use a created model that has a created real-time endpoint
        String mlModelId = "example-model-id";

        // Call GetMLModel to get the realtime endpoint URL
        GetMLModelRequest getMLModelRequest = new GetMLModelRequest();
        getMLModelRequest.setMLModelId(mlModelId);
        GetMLModelResult mlModelResult = client.getMLModel(getMLModelRequest);

        // Validate that the ML model is completed
        if (!mlModelResult.getStatus().equals(EntityStatus.COMPLETED.toString())) {
               System.out.println("ML Model is not completed: " +
 mlModelResult.getStatus()");
               return;
        }

        // Validate that the realtime endpoint is ready
        if (!
mlModelResult.getEndpointInfo().getEndpointStatus().equals(RealtimeEndpointStatus.READY.toString())
{
               System.out.println("Realtime endpoint is not ready: " +
 mlModelResult.getEndpointInfo().getEndpointStatus());
               return;
        }
```

Android - Kotlin

```
// Call GetMLModel to get the realtime endpoint URL
val modelRequest = new GetMLModelRequest()
modelRequest.mLModelID = "example-model-id"
val modelResult = client.getMLModel(modelRequest);

// Validate that the ML model is completed
if (modelResult.status != EntityStatus.COMPLETED.toString()) {
       Log.d(TAG, "ML Model is not completed: ${modelResult.status}");
       return;
}

// Validate that the realtime endpoint is ready
if (modelResult.endpointInfo.endpointStatus != RealtimeEndpointStatus.READY.toString())
 {
       Log.d(TAG, "Realtime endpoint is not ready:
 ${modelResult.endpointInfo.endpointStatus}");
       return;
}
```

Once the real-time endpoint is ready, we can begin calling Predict. Note that you must pass the real-time endpoint through the PredictRequest.

Android - Java

```
// Create a Predict request with your ML model ID and the appropriate Record mapping
PredictRequest predictRequest predictRequest = new PredictRequest();
predictRequest.setMLModelId(mlModelId);

HashMap<String, String> record = new HashMap<String, String>();
record.put("example attribute", "example value");

predictRequest.setRecord(record);
predictRequest.setPredictEndpoint(mlModelResult.getEndpointInfo().getEndpointUrl());
```

```
// Call Predict and print out your prediction
PredictResult predictResult = client.predict(predictRequest);
Log.d(LOG_TAG. predictResult.getPrediction());

// Do something with the prediction
// ...
```

Android - Kotlin

```
// Create a Predict request with your ML model ID and the appropriate Record mapping
val predictRequest predictRequest = PredictRequest().apply {
        mLModelID = "example-model-id"
        record = mapOf("example attribute" to "example value")
        predictEndpoint = modelResult.endpointInfo.getEndpointUrl
    }

    val predictResult = client.predict(predictRequest)
    Log.d(LOG_TAG, predictResult.prediction)

    // Do something with the prediction
    // ...
```

Additional Resources

- Developer Guide
- Service API Reference

iOS: Amazon Machine Learning

Amazon Machine Learning (ML) is a service that makes it easy for developers of all skill levels to use machine learning technology. The SDK for iOS provides a simple, high-level client designed to help you interface with Amazon Machine Learning service. The client enables you to call Amazon ML's real-time API to retrieve predictions from your models and enables you to build mobile applications that request and take actions on predictions. The client also enables you to retrieve the real-time prediction endpoint URLs for your ML models.

Integrate Amazon Machine Learning

To use the Amazon Machine Learning mobile client, you'll need to integrate the SDK for iOS into your app and import the necessary libraries. To do so, follow these steps:

#. Download the SDK and unzip it as described in Setup the SDK for iOS #. The instructions direct you to import the headers for the services you'll be using. For Amazon Machine Learning, you need the following import.

iOS - Swift

```
import AWSMachineLearning
```

iOS - Objective C

```
#import <AWSMachineLearning/AWSMachineLearning.h>
```

Configure Credentials

You can use Amazon Cognito to provide temporary AWS credentials to your application. These credentials let the app access your AWS resources. To create a credentials provider, follow the instructions at Providing AWS Credentials.

To use Amazon Machine Learning in an application, you must set the proper permissions. The following IAM policy allows the user to perform the actions shown in this tutorial on two actions identified by ARN.

```
{
    "Statement": [{
    "Effect": "Allow",
    "Action": [
        "machinelearning:GetMLModel",
        "machinelearning:Predict"
    ],
    "Resource": "arn:aws:machinelearning:use-east-1:11122233444:mlmodel/example-model-id"
    }]
}
```

This policy should be applied to roles assigned to the Amazon Cognito identity pool, but you will need to replace the Resource value with the correct account ID and ML Model ID. You can apply policies at the IAM console. To learn more about IAM policies, see Introduction to IAM.

Create an Amazon Machine Learning Client

Once you've imported the necessary libraries and have your credentials object, you can instantiate AWSMachineLearningGetMLModelInput.

iOS - Swift

```
let getMlModelInput = AWSMachineLearningGetMLModelInput()
```

Objective C

```
AWSMachineLearningGetMLModelInput *getMLModelInput = [AWSMachineLearningGetMLModelInput
 new];
```

Making a Predict Request

Prior to calling Predict, make sure you have not only a completed ML Model ID but also a created real-time endpoint for that ML Model ID. This cannot be done through the mobile SDK; you will have to use the Machine Learning Console or an alternate SDK. To validate that this ML can be used for real-time Predictions.

iOS - Swift

```
// Use a created model that has a created real-time endpoint
let mlModelId = "example-model-id";
// Call GetMLModel to get the realtime endpoint URL
let getMlModelInput = AWSMachineLearningGetMLModelInput()
getMlModelInput!.mlModelId = mlModelId;

machineLearning.getMLModel(getMlModelInput!).continueOnSuccessWith { (task) -> Any? in
    if let getMLModelOutput = task.result {

        if (getMLModelOutput.status != AWSMachineLearningEntityStatus.completed) {
```

```
                print("ML Model is not completed");
                return nil;
        }

        // Validate that the realtime endpoint is ready
        if (getMLModelOutput.endpointInfo!.endpointStatus !=
 AWSMachineLearningRealtimeEndpointStatus.ready) {
            print("Realtime endpoint is not ready");
            return nil;
        }
    }

return nil
}
```

Objective C

```
// Use a created model that has a created real-time endpoint
NSString *MLModelId = @"example-model-id";

// Call GetMLModel to get the realtime endpoint URL
AWSMachineLearningGetMLModelInput *getMLModelInput = [AWSMachineLearningGetMLModelInput
 new];
getMLModelInput.MLModelId = MLModelId;

[[[MachineLearning getMLModel:getMLModelInput] continueWithSuccessBlock:^id(AWSTask
 *task) {
    AWSMachineLearningGetMLModelOutput *getMLModelOutput = task.result;

    // Validate that the ML model is completed
    if (getMLModelOutput.status != AWSMachineLearningEntityStatusCompleted) {
        NSLog(@"ML Model is not completed");
        return nil;
    }

    // Validate that the realtime endpoint is ready
    if (getMLModelOutput.endpointInfo.endpointStatus !=
 AWSMachineLearningRealtimeEndpointStatusReady) {
        NSLog(@"Realtime endpoint is not ready");
        return nil;
    }
}
```

Once the real-time endpoint is ready, we can begin calling Predict. Note that you must pass the real-time endpoint through the PredictRequest.

iOS - Swift

```
// Create a Predict request with your ML Model id and the appropriate
let predictInput = AWSMachineLearningPredictInput()
predictInput!.predictEndpoint = getMLModelOutput.endpointInfo!.endpointUrl;
predictInput!.mlModelId = mlModelId;
predictInput!.record = record

return machineLearning.predict(predictInput!)
```

Objective C

```
// Create a Predict request with your ML Model id and the appropriate Record mapping.
AWSMachineLearningPredictInput *predictInput = [AWSMachineLearningPredictInput new];
predictInput.predictEndpoint = getMLModelOutput.endpointInfo.endpointUrl;
```

```
predictInput.MLModelId = MLModelId;
predictInput.record = @{};

// Call and return prediction
return [MachineLearning predict:predictInput];
```

Additional Resources

- Developer Guide
- API Reference

How To For Platform Specific Tasks

This section provides information on the steps for achieving specific tasks for integrating AWS Services into iOS apps.

Topics

iOS: Working with Asynchronous Tasks

To work with asynchronous operations without blocking the UI thread, the SDK provides two options:

- `completionHandler`, a streamlined class which offers a simple, common pattern for most API calls

and

- `AWSTask`, a class which is a renamed version of BFTask from the Bolts framework. AWSTasks gives advantages for more complex operations like chaining asynchronous requests.

  For complete documentation on Bolts, see the Bolts-ObjC repo.

Using completionHandler

Most simple asynchronous API method calls can use `completionHandler` to handle method callbacks. When an asynchronous method is complete, `completionHandler` returns two parts: a response object containing the method's return if the call was successful, or `nil` if failed; and an error object containing the `NSError` state when a call fails, or `nil` upon success.

Handling Asynchronous Method Returns with completionHandler

The following code shows typical usage of `completionHandler` using Amazon Kinesis Firehose as the example.

iOS - Swift

```
var firehose = AWSFirehose.default()
firehose.putRecord(AWSFirehosePutRecordInput(), completionHandler: {(_ response:
 AWSFirehosePutRecordOutput?, _ error: Error?) -> Void in
    if error != nil {
        //handle error
    }
```

```
    else {
        //handle response
    }
})
```

iOS - Objective-C

```
AWSFirehose *firehose = [AWSFirehose defaultFirehose];

[firehose putRecord:[AWSFirehosePutRecordInput new]
 completionHandler:^(AWSFirehosePutRecordOutput* _Nullable response, NSError *
 _Nullable error) {
       if(error){
        //handle error
       }else{
        //handle response
       }
    }];
```

Using AWSTask

An `AWSTask` object represents the result of an asynchronous method. Using `AWSTask`, you can wait for an asynchronous method to return a value, and then do something with that returned value. You can chain asynchronous requests instead of nesting them. This helps keep logic clean and code readable.

Handling Asynchronous Method Returns with AWSTask

The following code shows how to use `continueOnSuccessBlockWith:` and `continueWith:` to handle methods calls that return an `AWSTask` object.

iOS - Swift

```
let kinesisRecorder = AWSKinesisRecorder.default()

let testData = "test-data".data(using: .utf8)
kinesisRecorder?.saveRecord(testData, streamName: "test-stream-
name").continueOnSuccessWith(block: { (task:AWSTask<AnyObject>) -> AWSTask<AnyObject>?
 in
    // Guaranteed to happen after saveRecord has executed and completed successfully.
    return kinesisRecorder?.submitAllRecords()
}).continueWith(block: { (task:AWSTask<AnyObject>) -> Any? in
    if let error = task.error as? NSError {
        print("Error: \(error)")
        return nil
    }

    return nil
})
```

iOS - Objective-C

```
AWSKinesisRecorder *kinesisRecorder = [AWSKinesisRecorder defaultKinesisRecorder];

NSData *testData = [@"test-data" dataUsingEncoding:NSUTF8StringEncoding];
[[[kinesisRecorder saveRecord:testData
                   streamName:@"test-stream-name"] continueWithSuccessBlock:^id(AWSTask
 *task) {
    return [kinesisRecorder submitAllRecords];
}] continueWithBlock:^id(AWSTask *task) {
```

```
    if (task.error) {
        NSLog(@"Error: %@", task.error);
    }
    return nil;
}];
```

The `submitAllRecords` call is made within the `continueOnSuccessWith` / `continueWithSuccessBlock:` because we want to run `submitAllRecords` after `saveRecord:streamName:` successfully finishes running. The `continueWith` and `continueOnSuccessWith` won't run until the previous asynchronous call finishes. In this example, `submitAllRecords` is guaranteed to see the result of `saveRecord:streamName:`.

Handling Errors with AWSTask

The `continueWith:` and `continueOnSuccessWith:` block calls work in similar ways. Both ensure that the previous asynchronous method finishes executing before the subsequent block runs. However, they have one important difference: `continueOnSuccessWith:` is skipped if an error occurred in the previous operation, but `continueWith:` is always executed.

For example, consider the following scenarios, which refer to the preceding code snippet above.

- `saveRecord:streamName:` succeeded and `submitAllRecords` succeeded.

  In this scenario, the program flow proceeds as follows:

  1. `saveRecord:streamName:` is successfully executed.
  2. `continueOnSuccessWith:` is executed.
  3. `submitAllRecords` is successfully executed.
  4. `continueWith:` is executed.
  5. Because `task.error` is nil, it doesn't log an error.
  6. Done.
- `saveRecord:streamName:` succeeded and `submitAllRecords` failed.

  In this scenario, the program flow proceeds as follows:

  1. `saveRecord:streamName:` is successfully executed.
  2. `continueOnSuccessWith` is executed.
  3. `submitAllRecords` is executed with an error.
  4. `continueWithBlock:` is executed.
  5. Because `task.error` is NOT nil, it logs an error from `submitAllRecords`.
  6. Done.
- `saveRecord:streamName:` failed.

  In this scenario, the program flow proceeds as follows:

  1. `saveRecord:streamName:` is executed with an error.
  2. `continueOnSuccessWith:` is skipped and will NOT be executed.
  3. `continueWithBlock:` is executed.
  4. Because `task.error` is NOT nil, it logs an error from `saveRecord:streamName:`.
  5. Done.

Consolidated Error Logic with AWSTask

The preceding example consolidates error handling logic at the end of the execution chain for both methods called. It doesn't check for `task.error` in `continueOnSuccessBlockWith:`, but waits

until the `continueWith:` block executes to do so. An error from either the `submitAllRecords` or the `saveRecord:streamName:` method will be printed.

Per Method Error Logic with AWSTask

The following code shows how to implement the same behavior, but makes error handling specific to each method. `submitAllRecords` is only called if `saveRecord:streamName` succeeds, however, in this case, the `saveRecord:streamName` call uses `continueWith:`, the block logic checks `task.error` and returns nil upon error. If that block succeeds then `submitAllRecords` is called using `continueWith:` in a block that also checks `task.error` for its own context.

iOS - Swift

```
let kinesisRecorder = AWSKinesisRecorder.default()

let testData = "test-data".data(using: .utf8)
kinesisRecorder?.saveRecord(testData, streamName: "test-stream-
name").continueWith(block: { (task:AWSTask<AnyObject>) -> AWSTask<AnyObject>? in
    if let error = task.error as? NSError {
        print("Error from 'saveRecord:streamName:': \(error)")
        return nil
    }
    return kinesisRecorder?.submitAllRecords()
}).continueWith(block: { (task:AWSTask<AnyObject>) -> Any? in
    if let error = task.error as? NSError {
        print("Error from 'submitAllRecords': \(error)")
        return nil
    }

    return nil
})
```

iOS - Objective-C

```
AWSKinesisRecorder *kinesisRecorder = [AWSKinesisRecorder defaultKinesisRecorder];

NSData *testData = [@"test-data" dataUsingEncoding:NSUTF8StringEncoding];
[[[kinesisRecorder saveRecord:testData
    streamName:@"test-stream-name"] continueWithBlock:^id(AWSTask *task) {
    if (task.error) {
        NSLog(@"Error from 'saveRecord:streamName:': %@", task.error);
        return nil;
    }
    return [kinesisRecorder submitAllRecords];
}]continueWithBlock:^id(AWSTask *task) {
    if (task.error) {
          NSLog(@"Error from 'submitAllRecords': %@", task.error);
    }
    return nil;
}];
```

Returning AWSTask or nil

Remember to return either an `AWSTask` object or `nil` in every usage of `continueWith:` and `continueOnSuccessWith:`. In most cases, Xcode provides a warning if there is no valid return present, but in some cases an undefined error can occur.

Executing Multiple Tasks with AWSTask

If you want to execute a large number of operations, you have two options: executing in sequence or executing in parallel.

In Sequence

You can submit 100 records to an Amazon Kinesis stream in sequence as follows:

iOS - Swift

```
var task = AWSTask<AnyObject>(result: nil)

for i in 0...100 {
    task = task.continueOnSuccessWith(block: { (task:AWSTask<AnyObject>) ->
 AWSTask<AnyObject>? in
        return kinesisRecorder!.saveRecord(String(format: "TestString-%02d",
 i).data(using: .utf8), streamName: "YourStreamName")
    })
}

task.continueOnSuccessWith { (task:AWSTask<AnyObject>) -> AWSTask<AnyObject>? in
    return kinesisRecorder?.submitAllRecords()
}
```

iOS - Objective-C

```
AWSKinesisRecorder *kinesisRecorder = [AWSKinesisRecorder defaultKinesisRecorder];

AWSTask *task = [AWSTask taskWithResult:nil];
for (int32_t i = 0; i < 100; i++) {
    task = [task continueWithSuccessBlock:^id(AWSTask *task) {
        NSData *testData = [[NSString stringWithFormat:@"TestString-%02d", i]
 dataUsingEncoding:NSUTF8StringEncoding];
        return [kinesisRecorder saveRecord:testData
                                streamName:@"test-stream-name"];
    }];
}

[task continueWithSuccessBlock:^id(AWSTask *task) {
    return [kinesisRecorder submitAllRecords];
}];
```

In this case, the key is to concatenate a series of tasks by reassigning `task`.

iOS - Swift

```
task.continueOnSuccessWith { (task:AWSTask<AnyObject>) -> AWSTask<AnyObject>? in
```

iOS - Objective-C

```
task = [task continueWithSuccessBlock:^id(AWSTask *task) {
```

In Parallel

You can execute multiple methods in parallel by using `taskForCompletionOfAllTasks:` as follows.

iOS - Swift

```
var tasks = Array<AWSTask<AnyObject>>()
for i in 0...100 {
    tasks.append(kinesisRecorder!.saveRecord(String(format: "TestString-%02d",
 i).data(using: .utf8), streamName: "YourStreamName")!)
```

```
}

AWSTask(forCompletionOfAllTasks: tasks).continueOnSuccessWith(block:
 { (task:AWSTask<AnyObject>) -> AWSTask<AnyObject>? in
    return kinesisRecorder?.submitAllRecords()
}).continueWith(block: { (task:AWSTask<AnyObject>) -> Any? in
    if let error = task.error as? NSError {
        print("Error: \(error)")
        return nil
    }

    return nil
})
```

iOS - Objective-C

```
AWSKinesisRecorder *kinesisRecorder = [AWSKinesisRecorder defaultKinesisRecorder];

NSMutableArray *tasks = [NSMutableArray new];
for (int32_t i = 0; i < 100; i++) {
    NSData *testData = [[NSString stringWithFormat:@"TestString-%02d", i]
 dataUsingEncoding:NSUTF8StringEncoding];
    [tasks addObject:[kinesisRecorder saveRecord:testData
                                      streamName:@"test-stream-name"]];
}

[[AWSTask taskForCompletionOfAllTasks:tasks] continueWithSuccessBlock:^id(AWSTask
 *task) {
    return [kinesisRecorder submitAllRecords];
}];
```

In this example you create an instance of `NSMutableArray`, put all of our tasks in it, and then pass it to `taskForCompletionOfAllTasks:`, which is successful only when all of the tasks are successfully executed. This approach may be faster, but it may consume more system resources. Also, some AWS services, such as Amazon DynamoDB, throttle a large number of certain requests. Choose a sequential or parallel approach based on your use case.

Executing a Block on the Main Thread with AWSTask

By default, `continueWithBlock:` and `continueWithSuccessBlock:` are executed on a background thread. But in some cases (for example, updating a UI component based on the result of a service call), you need to execute an operation on the main thread. To execute an operation on the main thread, you can use Grand Central Dispatch or `AWSExecutor`.

Grand Central Dispatch

The following example shows the use of `dispatch_async(dispatch_get_main_queue(), ^{...});` to execute a block on the main thread. For error handling, it creates a `UIAlertView` on the main thread when record submission fails.

iOS - Swift

```
let kinesisRecorder = AWSKinesisRecorder.default()

let testData = "test-data".data(using: .utf8)
kinesisRecorder?.saveRecord(testData, streamName: "test-stream-
name").continueOnSuccessWith(block: { (task:AWSTask<AnyObject>) -> AWSTask<AnyObject>?
 in
    return kinesisRecorder?.submitAllRecords()
}).continueWith(block: { (task:AWSTask<AnyObject>) -> Any? in
```

```
    if let error = task.error as? NSError {
        DispatchQueue.main.async(execute: {
            let alertController = UIAlertView(title: "Error!", message:
 error.description, delegate: nil, cancelButtonTitle: "OK")
            alertController.show()
        })
        return nil
    }

    return nil
})
```

iOS - Objective-C

```
AWSKinesisRecorder *kinesisRecorder = [AWSKinesisRecorder defaultKinesisRecorder];

NSData *testData = [@"test-data" dataUsingEncoding:NSUTF8StringEncoding];
[[[kinesisRecorder saveRecord:testData
                   streamName:@"test-stream-name"] continueWithSuccessBlock:^id(AWSTask
 *task) {
    return [kinesisRecorder submitAllRecords];
}] continueWithBlock:^id(AWSTask *task) {
    if (task.error) {
        dispatch_async(dispatch_get_main_queue(), ^{
            UIAlertView *alertView =
                [[UIAlertView alloc] initWithTitle:@"Error!"
                                           message:[NSString stringWithFormat:@"Error:
 %@", task.error]
                                          delegate:nil
                                 cancelButtonTitle:@"OK"
                                 otherButtonTitles:nil];
            [alertView show];
        });
    }
    return nil;
}];
```

AWSExecutor

Another option is to use `AWSExecutor` as follows.

iOS - Swift

```
let kinesisRecorder = AWSKinesisRecorder.default()

let testData = "test-data".data(using: .utf8)
kinesisRecorder?.saveRecord(testData, streamName: "test-stream-
name").continueOnSuccessWith(block: { (task:AWSTask<AnyObject>) -> AWSTask<AnyObject>?
 in
    return kinesisRecorder?.submitAllRecords()
}).continueWith(executor: AWSExecutor.mainThread(), block: { (task:AWSTask<AnyObject>)
 -> Any? in
    if let error = task.error as? NSError {
        let alertController = UIAlertView(title: "Error!", message: error.description,
 delegate: nil, cancelButtonTitle: "OK")
        alertController.show()
        return nil
    }

    return nil
})
```

iOS - Objective-C

```
AWSKinesisRecorder *kinesisRecorder = [AWSKinesisRecorder defaultKinesisRecorder];

NSData *testData = [@"test-data" dataUsingEncoding:NSUTF8StringEncoding];
[[[kinesisRecorder saveRecord:testData streamName:@"test-stream-name"]
          continueWithSuccessBlock:^id(AWSTask *task) {
      return [kinesisRecorder submitAllRecords];
}] continueWithExecutor:[AWSExecutor mainThreadExecutor] withBlock:^id(AWSTask *task) {
    if (task.error) {
        UIAlertView *alertView =
            [[UIAlertView alloc] initWithTitle:@"Error!"
                    message:[NSString stringWithFormat:@"Error: %@", task.error]
                    delegate:nil
                    cancelButtonTitle:@"OK"
                    otherButtonTitles:nil];
        [alertView show];
    }
    return nil;
}];
```

In this case, `withBlock:` (Objective-C) or `block:` (Swift) is executed on the main thread.

iOS: Preparing Your App to Work with ATS

If you use the iOS 9 SDK (or Xcode 7) or later, the Apple App Transport Security (ATS) feature might impact how your apps interact with some AWS services.

If your app targeted for iOS 9+ attempts to connect to an AWS service endpoint that does not yet meet all the ATS requirements, the connection may fail. The following sections provide instructions to determine if your app is affected, and what steps to take to mitigate the impact of ATS on your app.

Diagnosing ATS Conflicts

If your app stops working after being upgraded to Xcode 7 or later and iOS 9 or later, follow these steps to determine if it affected by ATS.

1. Turn on verbose logging of the AWS Mobile SDK for iOS by calling the following line in the `–application:didFinishLaunchingWithOptions:` application delegate.

   iOS - Swift

   ```
   @UIApplicationMain
   class AppDelegate: UIResponder, UIApplicationDelegate {

       var window: UIWindow?
       func application(application: UIApplication, didFinishLaunchingWithOptions
       launchOptions: [UIApplicationLaunchOptionsKey: Any]?) -> Bool {
           ...
           AWSLogger.default().logLevel = .verbose
           ...

           return true
       }
   }
   ```

   iOS - Objective-C

   ```
   #import <AWSCore/AWSCore.h>
   ```

```
@implementation AppDelegate

- (BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
    ...
    [AWSLogger defaultLogger].logLevel = AWSLogLevelVerbose;
    ...

    return YES;
}

@end
```

2. Run your app and make a request to an AWS service.
3. Search your log output for "SSL". The message containing: "An SSL error has occurred and a secure connection to the server cannot be made" indicates that your app is affected by the ATS changes.

```
2015-10-06 11:39:13.402 DynamoDBSampleSwift[14467:303540] CFNetwork SSLHandshake failed
 (-9824)
2015-10-06 11:39:13.403 DynamoDBSampleSwift[14467:303540] NSURLSession/NSURLConnection
 HTTP load failed (kCFStreamErrorDomainSSL, -9824)
2015-10-06 11:39:13.569 DynamoDBSampleSwift[14467:303540] CFNetwork SSLHandshake failed
 (-9824)
2015-10-06 11:39:13.569 DynamoDBSampleSwift[14467:303540] NSURLSession/NSURLConnection
 HTTP load failed (kCFStreamErrorDomainSSL, -9824)
Error: Error Domain=NSURLErrorDomain Code=-1200 "An SSL error has occurred and a secure
 connection to the server cannot be made." UserInfo={_kCFStreamErrorCodeKey=-9824,
 NSLocalizedRecoverySuggestion=Would you like to connect to the server anyway?,
 NSUnderlyingError=0x7fca343012f0 {Error Domain=kCFErrorDomainCFNetwork
 Code=-1200 "(null)" UserInfo={_kCFStreamPropertySSLClientCertificateState=0,
 _kCFNetworkCFStreamSSLErrorOriginalValue=-9824, _kCFStreamErrorDomainKey=3,
 _kCFStreamErrorCodeKey=-9824}}, NSLocalizedDescription=An SSL error has occurred
 and a secure connection to the server cannot be made., NSErrorFailingURLKey=https://
dynamodb.us-east-1.amazonaws.com/, NSErrorFailingURLStringKey=https://dynamodb.us-
east-1.amazonaws.com/, _kCFStreamErrorDomainKey=3}
```

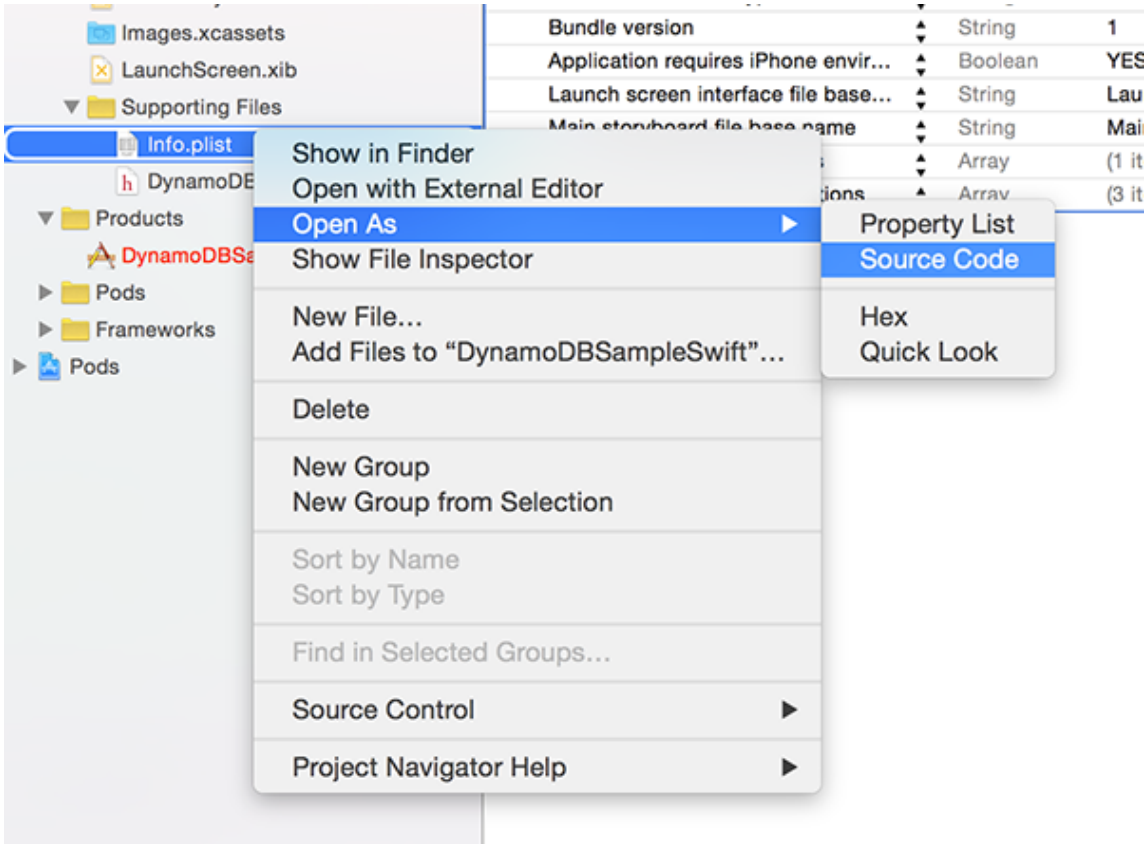
If you cannot find the SSL handshake error message, it is possible that another problem caused your app to stop working. Some internal behaviors change with major operating system updates, and it is common for previously unseen issues to surface.

If you are unable to resolve such issues, you can post code snippets, and steps to reproduce the issue on our forum or GitHub so that we can assist you in identifying the issue. Remember to include the versions of Xcode, iOS, and the AWS Mobile SDK.

Mitigating ATS Connection Issues

If you determine that your app is impacted by the diagnostic handshake error, you can configure the app to interact properly with the ATS feature by taking the following steps to add properties to your `Info.plist` file.

1. Locate your `Info.plist` and from the context menu select **Open As** > **Source Code**.



2. Copy and paste the following key as a direct child of the top level `<dict>` tag.

```
<plist version="1.0">
  <key>NSAppTransportSecurity</key>
  <dict>
      <key>NSExceptionDomains</key>
      <dict>
          <key>amazonaws.com</key>
          <dict>
                <key>NSThirdPartyExceptionMinimumTLSVersion</key>
                <string>TLSv1.0</string>
                <key>NSThirdPartyExceptionRequiresForwardSecrecy</key>
                <false/>
                <key>NSIncludesSubdomains</key>
                <true/>
          </dict>
          <key>amazonaws.com.cn</key>
          <dict>
                <key>NSThirdPartyExceptionMinimumTLSVersion</key>
                <string>TLSv1.0</string>
                <key>NSThirdPartyExceptionRequiresForwardSecrecy</key>
                <false/>
                <key>NSIncludesSubdomains</key>
                <true/>
          </dict>
      </dict>
  </dict>

  . . .
</plist>
```

After following these steps, your app should be able to access AWS endpoints while running on iOS 9 or later.

AWS Mobile Reference

Topics

Android and iOS API References

Android

- AWS Mobile SDK for Android API Reference
- Latest AWS Mobile SDK for Android Download
- AWS Mobile SDK for Android on GitHub
- AWS Mobile SDK for Android Samples

iOS

- AWS Mobile SDK for iOS API Reference
- AWS Mobile SDK for iOS on GitHub
- AWS Mobile SDK for iOS Samples

Amazon S3 Security Considerations for Mobile Hub Users

When you enable the Mobile Hub User File Storage or Hosting and Streaming features, 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. Hosting and Streaming also configures a CloudFront distribution that caches the assets stored in the bucket it creates. For the same type of information regarding the distribution, see cloudfront-security.

Access management

By default, access to Amazon S3 buckets and related objects are private: only the resource owner can access a bucket or assets contained in it. The administrator of a bucket can grant access that suits their design by attaching resource-based policies, such as bucket policy or access control lists (ACLs) to grant access to users or groups of users.

The Amazon S3 configuration provisioned by the AWS Mobile Hub Hosting and Streaming (p. 342) feature is example of setting bucket policy to a allow access to all users. This access policy makes sense in the context of publicly hosting a web app through this feature. We recommend, if it meets app design criteria, that developers also add the User Sign-in (p. 348) feature so that only authenticated users have access to an app's AWS resources like buckets and database.

For more information, see Managing Access Permissions to Your Amazon S3 Resources in the *Amazon S3 Developer Guide*.

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 S3 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 S3 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 S3 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 S3 User Guide*.

Amazon CloudFront Security Considerations for Mobile Hub Users

When you enable the AWS Mobile Hub Hosting and Streaming (p. 342) feature, an Amazon CloudFront distribution is created in your account. The distribution caches the web assets you store within an associated Amazon S3 bucket throughout a global network of Amazon edge servers. This provides your customers with fast local access to the web assets.

This topic describes the key CloudFront security-related features that you might want to use for your distribution. For the same type of information regarding the source bucket, see s3-security.

Access management

Hosting and Streaming makes assets in a distribution publically available. While this is the normal security policy for Internet based resources, you should consider restricting access to the assets if this is not the case. The best practice for security is to follow a ?minimal permissions? model and restrict access to resources as much as possible. You may want to modify resource-based policies, such as the distribution policy or access control lists (ACLs), to grant access only to some users or groups of users.

To protect access to any AWS resources associated with a Hosting and Streaming web app, such as buckets and database tables, we recommend restricting access to only authenticated users. You can add this restriction to your Mobile Hub project by enabling the User Sign-in (p. 348) feature, with the sign-in required option.

For more information, see Authentication and Access Control for CloudFront in the *Amazon CloudFront Developer Guide*.

Requiring the HTTPS Protocol

CloudFront supports use of the HTTPS protocol to encrypt communications to and from a distribution. This highly recommended practice protects both the user and the service. CloudFront enables you to require HTTPS both between customers and your distribution endpoints, and CloudFront between your distribution's caches and the source bucket where your assets originate. Global redirection of HTTP traffic to HTTPS, use of HTTPS for custom domains and other options are also supported.

For more information, see Using HTTPS with CloudFront in the *Amazon CloudFront Developer Guide*.

Securing Private Content

CloudFront supports a range of methods for protecting private content in a distribution cache. These include the use of signed cookies and signed URLs to restrict access to authenticated, authorized users.

A best practice is to use techniques like these on both the connection between the user and the distribution endpoint and between the distribution and the content Amazon S3 source bucket.

For more information, see the Serving Private Content through CloudFront section in the *Amazon CloudFront Developer Guide*.

Distribution Access Logging

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

For more information, see Access Logs for CloudFront in the *Amazon CloudFront Developer Guide*.

AWS Mobile Reference

Topics

AWS Identity and Access Management Usage in AWS Mobile Hub

Note
In depth understanding of AWS IAM, authentication, and access controls are not required to configure a backend for your mobile app using Mobile Hub.

- Control Access to Mobile Hub Projects (p. 314) - learn how to grant permissions for configuration of your Mobile Hub project.
- Mobile Hub Project Permissions Model (p. 313) - learn more about permissions you give Mobile Hub to configure AWS resources and services, see .
- IAM Authentication and Access Control for Mobile Hub (p. 317) - learn details of IAM and AWS authentication and access controls.

Mobile Hub Project Permissions Model

Important	To modify Mobile Hub projects in an account, a user must be granted administrative permissions (p. 315) by an account Administrator. Read this section for more information. If you are a user who needs additional permissions for a project, contact an administrator for the AWS account. For help with any issues related to the new permissions model, contact aws-mobilehub-customer@amazon.com.

Topics

- Mobile Hub Permissions Model (p. 313)
- What if I Currently Use MobileHub_Service_Role to Grant Mobile Hub Permissions? (p. 314)
- Why Did the Permissions Model Change? (p. 314)

Mobile Hub Permissions Model

Currently, Mobile Hub's permissions model uses the user's permissions directly when they perform operations in the Mobile Hub console or command line interface (p. 386). This model provides account administrators fine-grained access control over what operations their users can perform in the account, regardless of whether they are using Mobile Hub or they're using the console or command line interface to interact with services directly.

In order to modify projects, users are required to have permissions to use Mobile Hub (granted by AWSMobileHubFullAccess IAM policy), and they must have permission to perform whatever actions Mobile Hub takes on their behalf. In almost every case, this means an account administrator must grant the user the AdministratorAccess policy (p. 315) in order to provide access to the AWS resources Mobile Hub modifies. This is because, as project settings are modified, Mobile Hub will modify the IAM roles and policies used to enable the features affected by those settings. Changing IAM roles and policies allows the user to control access to resources in the account, and so they must have administrative permissions.

When an administrator does not want to grant administrative permissions for the full account, they can choose instead to provide each user or team their own sub-account using AWS Organizations (p. 316). Within their sub-account, a user will have full administrative permissions. Sub-account owners are only

limited in what they can do by the policy put in place by their administrator, and billing rolls up to the parent account.

What if I Currently Use MobileHub_Service_Role to Grant Mobile Hub Permissions?

Previously, Mobile Hub assumed a service role called `MobileHub_Service_Role` in order to modify service configurations on your behalf using the following managed policy:

https://console.aws.amazon.com/iam/home?#/policies/arn:aws:iam::aws:policy/service-role/AWSMobileHub_ServiceUseOnly

In that older model, all that was required to modify Mobile Hub projects was permissions to call Mobile Hub APIs, through the console or command line. An administrator could delegate those permissions by attaching the `AWSMobileHub_FullAccess` policy to an AWS IAM user, group, or role.

If the account of your Mobile Hub projects relies on the old model, the impact on those who are not granted AdministratorAccess permissions will be as follows.

- IAM users, groups and roles that have the `AWSMobileHub_FullAccess` policy will no longer have sufficient permissions to perform any mutating operations in Mobile Hub, either via the console or `awsmobile` command line interface (CLI).
- In order for IAM users, groups, or roles to be able to perform mutating operations using Mobile Hub, they must have the appropriate permissions. The two choices for an administrator to grant users permission (p. 314) to invoke all available operations in Mobile Hub are: attach the `AdministratorAccess` policy to the user, or a role they are attached to, or a group they are a member of; or alternatively, to use AWS Organizations to manage permissions.

Why Did the Permissions Model Change?

AWS Mobile Hub creates IAM roles and assigns them permissions in order to enable use of AWS resources in mobile apps. Such operations are considered administrative because they include enabling permission to perform operations on resources in the account. Previously, Mobile Hub's service role provided users who have been granted `AWSMobileHub_FullAccess` permissions with a path to escalate their own privileges to act on resources, potentially in ways their administrator did not intend to permit. Removing the service role, removes the path to escalate privileges and puts control of user permissions directly in the hands of the administrator for a Mobile Hub project.

Control Access to Mobile Hub Projects

Overview

This section describes two different ways to control access to your Mobile Hub projects:

- Grant a user administrative account permissions (p. 315)

  For individual developers, or groups whose requirements for segmenting access to their Mobile Hub projects are simple, permission can be granted by attaching the managed AdministratorAccess (p. 317) or AWSMobileHub_ReadOnly (p. 317) AWS managed policies to a user, a role they are attached to, or a group they belong to.

Or:

- Use AWS Organizations to manage permissions (p. 316)

  For organizations that require fine-grained access control and cost tracking for their Mobile Hub projects, AWS account administrators can provide sub-accounts and determine the policies that apply to their users.

To understand how Mobile Hub uses IAM policies attached to a user to create and modify services on a users behalf, see Mobile Hub Project Permissions Model (p. 313).

To understand AWS Identity and Access Management (IAM) in more detail, see IAM Authentication and Access Control for Mobile Hub (p. 317) and IAM Authentication and Access Control for Mobile Hub (p. 317).

Best Practice: Create IAM Users to Access AWS

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 in your AWS account, or use an existing IAM user, and then access Mobile Hub with that user. For more information, see AWS Security Credentials in the AWS General Reference.

You can create an IAM user for yourself or a delegate user using the IAM console. First, create an IAM administrator group, then create and assign a new IAM user to that group.

> **Note**
> Before any IAM user within an account can create a mobile Hub project, a user with administrative privileges for the account must navigate to the Mobile Hub console and create an initial project. This step provides confirmation that Mobile Hub can manage AWS services on your behalf.
> To learn more about assigning access rights to IAM users or groups, see IAM Authentication and Access Control for Mobile Hub (p. 317).

Grant Users Permissions to Mobile Hub Projects

Topics

Use the following steps to create a group and/or users, and grant users access to your Mobile Hub projects.

To grant permissions to a role, see Adding Permissions in the *AWS IAM User Guide*.

Create a New IAM User in Your Account and Grant Mobile Hub Permissions

1. Open the IAM console. On the left, choose **Users**, and then choose **Add User**.
2. Type a user name, select the checkboxes for **Programmatic access** and **AWS Management Console access**.
3. Choose the password policy you prefer. Then choose **Next: Permissions**.
4. In the **Add user to group** tab, select the **Administrators** or **Read_Only** group for the user, and choose **Next, Review**.

   In the process, you will see options to customize the user's password, alert them about their new account via email, and to download their access key ID, key value and password.
5. Choose **Create user**.
6. To apply policy:
   - If you have created a group to manage project permissions, choose **Add user to group**, select the group, choose **Next: Review**, then choose choose **Create User**.

   Or:

- If you are managing project permissions per user, choose **Attach existing policies directly**, select the policy you want to attach, **AdministratorAccess** or **AWSMobileHub_ReadOnly**, and then choose **Create user**.

Create an IAM Group

1. Sign in to the AWS Management Console and open the IAM console at http://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` or `Read_Only`, and then choose **Next Step**.
4. In the list of policies, select the check box next to the **AdministratorAccess** policy to grant full permissions to the group, or **AWSMobileHub_ReadOnly** to grant only read access. 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**.

Grant Mobile Hub Permissions to an Existing Account User

1. On the left, choose **Policies**.
2. Choose the link for the managed policy, **AdministratorAccess** or **AWSMobileHub_ReadOnly** you want to attach.
3. Choose **Attached Entities**.
4. Choose **Attach**.
5. Choose the users, roles, or groups you want to grant permissions.
6. Choose **Attach Policy**.

Use AWS Organizations to Manage Permissions

AWS Organizations can be used to manage permissions for groups that need to segment access to their Mobile Hub projects. For example, an administrator could provide an account for each developer on a team. Within their own account, each user would have the permissions granted by the administrator. The steps to acheive this would be:

1. If you do not have an AWS account, sign up for the AWS Free Tier.
2. Create an organization in the AWS Organizations console.
3. Create or add existing accounts for each user in the organization.
4. Invite the users.
5. Create a organizational unit for the developers.
6. Enable and attach a policy for members of the unit.

   The policy you attach will apply within the scope of the AWS account of a user. You may want to limit access to services and capabilites not required for Mobile Hub use. For instance, the following policy, grants all permissions defined in the `FullAWSAccess` managed policy, but excludes access to the Amazon EC2 service.

```
"Statement": [
        {
            "Effect": "Allow",
            "Action": "*",
            "Resource": "*"
        },
        {
            "Effect": “Deny”,
```

```
            "Action": “ec2:*”,
            "Resource": "*"
        }
]
```

For step by step instructions, see the tutorial at Creating and Managing an AWS Organization.

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.

- **AdministratorAccess**

  This policy provides unlimited access to AWS services in the account. That includes 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/AdministratorAccess$jsonEditor

- **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.
  - http://console.aws.amazon.com/iam/home?region=us-east-1#policies/arn:aws:iam::aws:policy/AWSMobileHub_ReadOnly

  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 **AdminstratorAccess** policy have those permissions. If you need additional permissions in Mobile Hub, please contact your administrator and request the **AdminstratorAccess** policy.

IAM Authentication and Access Control for Mobile Hub

> **Note**
> *In depth understanding of AWS IAM, authentication, and access controls are not required to configure a backend for your mobile app using Mobile Hub.*

Mobile Hub uses AWS credentials and permissions policies to allow a user to view and/or create and configure the back-end features the user selects for their 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

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. 319) 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.

- Control Access to Mobile Hub Projects (p. 314)

Overview of Access Permissions Management for Mobile Hub Projects

> **Note**
> *In depth understanding of AWS IAM, authentication, and access controls are not required to configure a backend for your mobile app using Mobile Hub.*

Every AWS resource is owned by an AWS account. Permissions to view, create, and/or access the resources (p. 314) are governed by policies.

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. 319)
- Managing Access to Resources (p. 320)
- Specifying Policy Elements: Actions, Effects, Resources, and Principals (p. 321)

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 polices) and policies attached to a resource are referred to as resource-based policies.

Topics

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 :ref: reference-mobile-hub-project-permissions-model `.

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:

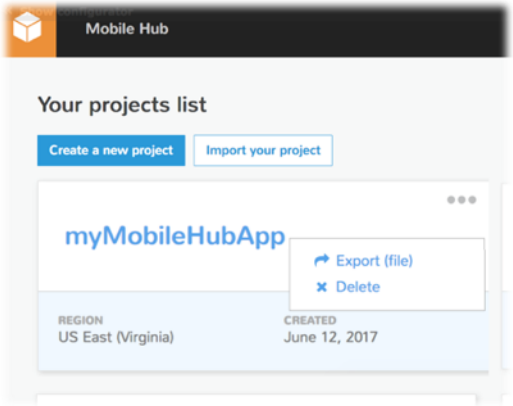
- **Resource** - In a policy, you use an Amazon Resource Name (ARN) to identify the resource to which the policy applies.
- **Action** - You use action keywords to identify resource operations that you want to allow or deny. For example, the `s3:Createbucket` permission allows Mobile Hub to perform the 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).

Exporting and Importing AWS Mobile Hub Projects

Overview

Mobile Hub provides the ability to export and import YAML files that describe the configuration of your Mobile Hub project. Anyone with an AWS account can import an exported project configuration file to deploy a new project, with new AWS resources that match the configuration being imported.

This feature enables you to replicate the AWS service configuration of an exported project. While the data in a project's tables is not exported, files in storage or hosting buckets and API handler function code can be manually added to your exported project definition. To learn more, see import-export-manual.



To export a project configuration file

1. Navigate to your project list in the Mobile Hub console.
2. Hover over the ellipses (three dots) in the upper right of the project card.
3. Choose **Export (file)** in the upper right of the card for the project you want to export.
4. Save your project export file.

To learn more about the content of an exported project configuration file, see Structure of a Project Export .yml File (p. 325).

To import a project

1. Navigate to your project list in the Mobile Hub console.
2. Choose **Import your project** in the upper left of the page.
3. Browse or drag a project definition file into the **Import project configuration** dialog.
4. Choose **Import project**.

Sharing Your Project Configuration with a Deploy to AWS Mobile Hub Link

In any public GitHub repo, you can provide a link that instantly kicks off creation of a new Mobile Hub project by importing the exported project configuration file define in the link's querystring. The form of the link should be:

```
https://console.aws.amazon.com/mobilehub/home?#/?config=YOUR-MOBILE-HUB-
PROJECT-CONFIGURATION-LOCATION
```

For example, the following HTML creates a link that provides instant configuration of an app's AWS backend services, based on Mobile Hub features defined in `react-sample.zip`. To see this code in action, see `README.md` for the AWS Mobile React Sample.

```
<p align="center">
   <a target="_blank" href="https://console.aws.amazon.com/mobilehub/home?#/?
config=https://github.com/awslabs/aws-mobile-react-sample/blob/master/backend/
import_mobilehub/react-sample.zip">
   <span>
       <img height="100%" src="https://s3.amazonaws.com/deploytomh/button-deploy-aws-
mh.png"/>
   </span>
   </a>
```

```
</p>
```

The querystring portion of the link can point to the location of a Mobile Hub project configuration `mobile-hub-project.yml` file or a project export `.zip` file containing a `mobile-hub-project.yml` file.

> **Important**
> If you are using a `.zip` file it must conform to the structure and content required by a Mobile Hub project configuration import. For details, see Structure of a Project Export .zip File (p. 325).

Limitations of Importing Projects

Topics

Maximum Project Definition File Size is 10MB

Import of Mobile Hub project `.zip` or `.yml` files larger than 10MB is not supported.

Project Components that Require Manual Export

To enable import of the following project configuration items, you must manually modify your project's exported `.zip` file:

- Data User Storage Contents

  To import files stored in a User File Storage Amazon S3 bucket in your original project, see Importing User File Storage Contents (p. 327).
- Hosting and Streaming Contents

  To import files hosted in a Hosting and Streaming bucket in your original project, see Importing Hosting and Streaming Contents (p. 328).
- SAML Federation

  To import User Sign-in SAML federation configuration from your original project, see Importing SAML Federated User Sign-in (p. 329).
- Cloud Logic API Handlers

  To import Cloud Logic API handler code and configuration from your original project, see Importing API Handlers for Cloud Logic APIs (p. 329).

  > **Note**
  > Calling Cloud Logic APIs from a browser requires that Cross-Origin Resource Sharing (CORS) is configured for each API path. To enable CORS configuration when your project is imported, see Importing Cross-Origin Resource Sharing (CORS) Configuration (p. 331).

Cross Account Credentials

Some features require credentials and assets that are associated with the AWS account where they are configured. Mobile Hub projects that contain such features can only be imported into the account that exported them. Features with this restriction include Cloud Logic APIs that were created outside of the

Mobile Hub project being exported, messaging provider credentials for Push Notification, and Amazon SNS topics.

Mobile Hub Feature	Can be exported from one AWS account and imported into another?
User Sign-in	Yes
NoSQL Database	Yes
Cloud Logic	Using APIs created within your Mobile Hub project: Yes Using APIs imported into your project: No (for remedy, see Cannot Import an API (p. 361))
User File Storage	Yes
App Content Delivery	Yes
Connectors	Yes
Push Notifications	No (for remedy, see Cannot Import Push Credentials (p. 362))
Messaging and Analytics (Push Notification)	No (for remedy, see Cannot Import Push Credentials (p. 362))

Project Components that Are Not Exported

The following items are not supported by the Mobile Hub import/export feature:

- Custom policy

  When you enable a Mobile Hub feature, a set of AWS services is deployed. Mobile Hub attaches default access roles and policies to these objects. When a project is imported, the default roles and policies are applied.

  In your original project, you can to modify or add to these defaults; for example, to set access to a data table to read only. When you export your project configuration, any such customizations are not included in the project export. To enable your custom policy in an imported project, the importer must manually configure those policies in the imported project. In addition to your project export file, we recommend you provide both your policy JSON and step by step instructions for importers. These instructions should describe how to use AWS consoles or the AWS CLI to implement your customizations.
- Legacy Cloud Logic

  Import and export are not supported for projects using the legacy Cloud Logic feature. A project of this kind calls Lambda functions directly. The current version of Cloud Logic makes RESTful calls to Amazon API Gateway APIs linked to Lambda function handlers.

Mobile Hub Project Export Format

AWS Mobile Hub provides the ability to export a YAML file containing the configuration of your project. The YAML file itself can be imported or it can be included in a `.zip` file with other project components

that get deployed during project import. This section describes the anatomy of the YAML and a typical Mobile Hub project export `.zip` file. For more information about the Mobile Hub Import/Export feature, see Exporting and Importing AWS Mobile Hub Projects (p. 321).

Topics

Structure of a Project Export .zip File

When you choose **Export (file)**, Mobile Hub generates a `.zip` file named for your project.

Default file structure

Mobile Hub also generates a `mobile-hub-project.yml` project configuration file in the `.zip` root. A valid `mobile-hub-project.yml` file in this location is required for Mobile Hub project import to succeed.

Example file structure

File structure of the `.zip` file an exported project, configured to include deployment of both SAML federation and Cloud Logic API handlers, is as follows:

- */your-project-name*`.zip`
  - `mobile-hub-project.yml`
  - `saml.xml`
  - `lambda API handler functions`
  - `user data stored files`
  - `hosted files`

Files in a project export `.zip` file can be arranged in folders. The relative paths within the archive must be reflected in the project definition YAML key values that refer to their paths.

> **Note**
> The presence of any files or folders in the project configuration `.zip` file, other than those described in the preceding section, may be ignored or cause issues upon import.

Structure of a Project Export .yml File

In the abstract, the basic structure of a Mobile Hub project export `.yml` file is as follows:

```
features:
    FEATURE-TYPE: !com.amazonaws.mobilehub.v0.:FEATURE-TYPE
          components:
            FEATURE-NAME: !com.amazonaws.mobilehub.v0.FEATURE-TYPE
                attributes:
                    ATTRIBUTE-NAME: !com.amazonaws.mobilehub.v0.ATTRIBUTE-VALUE
                OTHER-FEATURE-PROPERTY-TYPES: OTHER-FEATURE-PROPERTY-VALUES
            . . .
```

The following YAML is a sample of the `mobile-hub-project.yml` exported from a project with many Mobile Hub features enabled. The project definition has also been manually updated to enable the import and upload of components of the original project. These components include files stored in the original project's User File Storage bucket, files hosted in its Hosting and Streaming bucket, and API handler code in its Lambda functions.

```
--- !com.amazonaws.mobilehub.v0.Project
features:
```

```
cloudlogic: !com.amazonaws.mobilehub.v0.CloudLogic
  components:
    api-name: !com.amazonaws.mobilehub.v0.API
      attributes:
        name: api-name
        requires-signin: true
        sdk-generation-stage-name: Development
      paths:
        /items: !com.amazonaws.mobilehub.v0.Function
          codeFilename: uploads/lambda-archive.zip
          description: "Handler for calls to resource path : /items"
          enableCORS: true
          handler: lambda.handler
          memorySize: "128"
          name: handler-name
          runtime: nodejs6.10
          timeout: "3"
        "/items/{proxy+}": !com.amazonaws.mobilehub.v0.Function
          codeFilename: uploads/lambda-archive.zip
          description: "Handler for calls to resource path : /items/{proxy+}"
          enableCORS: true
          handler: lambda.handler
          memorySize: "128"
          name: handler-name
          runtime: nodejs6.10
          timeout: "3"
content-delivery: !com.amazonaws.mobilehub.v0.ContentDelivery
  attributes:
    enabled: true
    visibility: public-global
  components:
    release: !com.amazonaws.mobilehub.v0.Bucket {}
database: !com.amazonaws.mobilehub.v0.Database
  components:
    database-nosql: !com.amazonaws.mobilehub.v0.NoSQLDatabase
      tables:
        - !com.amazonaws.mobilehub.v0.NoSQLTable
          attributes:
          id: S
          hashKeyName: id
          hashKeyType: S
          rangeKeyName: ""
          rangeKeyType: ""
          tableName: ___DYNAMIC_PREFIX___-bbq-order
          tablePrivacy: public
        - !com.amazonaws.mobilehub.v0.NoSQLTable
          attributes:
          id: S
          hashKeyName: id
          hashKeyType: S
          rangeKeyName: ""
          rangeKeyType: ""
          tableName: ___DYNAMIC_PREFIX___-bbq_restaurants
          tablePrivacy: public
        - !com.amazonaws.mobilehub.v0.NoSQLTable
          attributes:
          id: S
          restaurant_id: S
          hashKeyName: restaurant_id
          hashKeyType: S
          rangeKeyName: id
          rangeKeyType: S
          tableName: ___DYNAMIC_PREFIX___-bbq_menu_item
          tablePrivacy: public
sign-in: !com.amazonaws.mobilehub.v0.SignIn
  attributes:
```

```
      enabled: true
      optional-sign-in: false
    components:
      sign-in-user-pools: !com.amazonaws.mobilehub.v0.UserPoolsIdentityProvider
        attributes:
          alias-attributes:
            - email
            - phone_number
          mfa-configuration: ON
          name: userpool
          password-policy: !com.amazonaws.mobilehub.ConvertibleMap
            min-length: "8"
            require-lower-case: true
            require-numbers: true
            require-symbols: true
            require-upper-case: true
  user-files: !com.amazonaws.mobilehub.v0.UserFiles
    attributes:
      enabled: true
  user-profiles: !com.amazonaws.mobilehub.v0.UserSettings
    attributes:
      enabled: truename: myProject
region: us-east-1
uploads:
    - !com.amazonaws.mobilehub.v0.Upload
      fileName: stored-file
      targetS3Bucket: user-file.png
    - !com.amazonaws.mobilehub.v0.Upload
      fileName: hosted-file
      targetS3Bucket: hosting.html
    - !com.amazonaws.mobilehub.v0.Upload
      fileName: api-handler-file.zip
      targetS3Bucket: deployments
```

Manually Exported Project Components

This section describes how to manually add project components to an exported project definition.

Topics

Importing User File Storage Contents

When a project that enables User File Storage is exported, files stored in its Amazon S3 bucket are not included in its exported project definition. You can manually configure the project definition to upload those files to the new bucket of the imported project.

To configure import and upload of project files stored in a User File Storage bucket

1. Uncompress your exported project `.zip` file.
2. Copy and paste each file that you want uploaded during import into the unzipped file folder.
3. Add file paths to your exported project definition:
   a. Open the `mobile-hub-project.yml` file of the export in an editor.
   b. If not already present, create an `uploads:` node at the root level.
   c. For each file to be uploaded, add the following three items under `uploads:`.

i. The namespace – `!com.amazonaws.mobilehub.v0.Upload`
ii. The key `fileName:` with the value of the path to the file within the project definition `.zip` file.
iii. The key `targetS3Bucket:` with the value of `user-files`.

```
--- !com.amazonaws.mobilehub.v0.Project
features:
  sign-in: !com.amazonaws.mobilehub.v0.SignIn {}
  user-files: !com.amazonaws.mobilehub.v0.UserFiles
    attributes:
      enabled: true
  user-profiles: !com.amazonaws.mobilehub.v0.UserSettings
    attributes:
      enabled: true
name: userfiles
region: us-east-1
uploads:
  - !com.amazonaws.mobilehub.v0.Upload
    fileName: {example1.png}
    targetS3Bucket: user-files
  - !com.amazonaws.mobilehub.v0.Upload
    fileName: {example2.xml}
    targetS3Bucket: user-files
. . .
```

4. Rezip the files within the uncompressed project definition file (not the folder containing those files, because that causes a path error).

Importing Hosting and Streaming Contents

When a project that enables Hosting and Streaming is exported, files stored in its Amazon S3 bucket are not included in the exported project definition. You can manually configure the project definition to upload those files to the new bucket of the imported project.

To configure import and upload of project files stored in a Hosting and Streaming bucket

1. Uncompress your exported project `.zip` file.
2. Copy and paste each file that you want uploaded during import into the unzipped file folder.
3. Add file paths to your exported project definition:
   a. Open the `mobile-hub-project.yml` file of the export in an editor.
   b. If not already present, create an `uploads:` node at the root level.
   c. For each file to be uploaded, add the following three items under `uploads:`.
      i. The namespace – `!com.amazonaws.mobilehub.v0.Upload`
      ii. The key `fileName:` with the value of the path to the file within the project definition `.zip` file.
      iii. The key `targetS3Bucket:` with the value of `hosting`.

```
--- !com.amazonaws.mobilehub.v0.Project
features:
  content-delivery: !com.amazonaws.mobilehub.v0.ContentDelivery
    attributes:
      enabled: true
      visibility: public-global
    components:
      release: !com.amazonaws.mobilehub.v0.Bucket {}

. . .

uploads:
  - !com.amazonaws.mobilehub.v0.Upload
```

```
    fileName: {example1.html}
    targetS3Bucket: hosting
  - !com.amazonaws.mobilehub.v0.Upload
    fileName: {example2.js}
    targetS3Bucket: hosting
. . .
```

4. Rezip the files within the uncompressed project definition file (not the folder containing those files, because that causes a path error).

Importing SAML Federated User Sign-in

Configuring SAML federation for the Mobile Hub User Sign-in feature requires you to supply the SAML XML configuration (`saml.xml`) of the identity provider you federate. The SAML XML configuration is not included in the `.zip` file exported by Mobile Hub.

To configure an exported project to deploy the original project's SAML federation when it is imported

1. Uncompress your exported project `.zip` file.
2. Copy your identity provider's `saml.xml` file into the root folder of the uncompressed `.zip` file.
3. Rezip the files within the uncompressed project definition file (not the folder containing those files, because that causes a path error).

Importing API Handlers for Cloud Logic APIs

The Mobile Hub Cloud Logic feature pairs a RESTful API surface (API Gateway) with serverless API handler functions (Lambda). While Mobile Hub supports exporting and importing the definitions of API and handler objects that Cloud Logic configures, the API handler function code is not exported.

Mobile Hub enables you to manually configure your project export `.zip` file to deploy your API handler function code as part of the project import when the following conditions are met:

- Your API handler accesses only DynamoDB tables. Import of API handlers that access other AWS services, such as Amazon S3, is not currently supported.
- Your handler code is factored to use Lambda environmental variables to refer to those DynamoDB tables.

  When Mobile Hub imports API handler code, it uses environmental variables to map data operations to the new tables created by the import. You can define the key name of environmental variables in the project's definition YAML to match constant names you define in the project's Lambda API handler function code. The following example shows a Lambda function constant being equated to an environmental variable.

```
const YOUR-FUNCTION-CONSTANT-NAME = process.env.KEY-NAME-DEFINED-IN-YAML;";

// example
const MENU_TABLE_NAME = process.env.MENU_TABLE_NAME;
```

  The steps that follow these notes describe how to define your environmental variables in project definition YAML.

  Note
  An alternative is to use the `MOBILE_HUB_DYNAMIC_PREFIX` project identifier prefix that Mobile Hub generates. Mobile Hub configures its value to be the unique identifier for the imported project. When you append a valid table name to that prefix in your function code, it composes a valid identifier for the table in the imported project. The following example shows a Lambda function constant being equated to an environmental variable.

```
const YOUR-FUNCTION-CONSTANT-NAME = process.env.MOBILE_HUB_DYNAMIC_PREFIX + "-
YOUR-TABLE-NAME";

// example
const MENU_TABLE_NAME = process.env.MOBILE_HUB_DYNAMIC_PREFIX + "-bbq-menu";
```

This method does not require additional manual configuration of the project definition YAML.

The AWS Mobile React sample app provides an end to end example of using environmental variables to access data tables through an API and its handler. Take the following steps for each API handler whose code you want to import. Examples from the sample app are given in line.

To enable import of |LAM| handler functions for your exported Cloud Logic API

1. Uncompress your exported project `.zip` file.
2. Copy your Lambda function(s) into the uncompressed file.
   a. Go to the Amazon S3 console and search for your Mobile Hub project name.
   b. Choose the bucket with the name containing `-deployments-`, then choose the `uploads` folder.
   c. Copy and save the name(s) of the Lambda function file(s) in the folder for use in following steps.
   d. Copy the Lambda function file(s) in the folder into your unzipped exported project file.
3. Add file paths to your exported project definition.
   a. Open the `mobile-hub-project.yml` file of the export in an editor.
   b. If not already present, create an `uploads:` node at the root level.
   c. For each file to be uploaded, add the following three items under `uploads:`.
      i. The namespace – `!com.amazonaws.mobilehub.v0.Upload`
      ii. The key `fileName:` with the value of the path to the file within the project definition `.zip` file.
      iii. The key `targetS3Bucket:` with the value of `deployments`.
   d. If not already present in each Cloud Logic `. . . paths: items` node, create a `codeFilename:` key with the value of the path of the Lambda function code file for that handler.

      Note
      The path in this case is relative to the root of the `-deployments-`Amazon S3 bucket Mobile Hub provisioned for Cloud Logic. Typically, Mobile Hub places these files in an `/uploads` folder.
      If no `codeFilename` is specified, then Mobile Hub deploys a default handler that echos requests it receives.
   e. Add environmental variables to your exported project definition.

      For each Cloud Logic `. . . paths: items` node that describes a handler that interacts with a DynamoDB table, add an `environment:` node with child members that are composed by concatenating an environmental variable name, with the string `__DYNAMIC_PREFIX__`, and the associated table name. The variable name should map to the associated variable in your Lambda API handler function code.

```
--- !com.amazonaws.mobilehub.v0.Project
features:
  cloudlogic: !com.amazonaws.mobilehub.v0.CloudLogic
    components:
      api-name: !com.amazonaws.mobilehub.v0.API
        attributes:
          name: api-name
          requires-signin: true
          sdk-generation-stage-name: Development
        paths:
```

```
          /items: !com.amazonaws.mobilehub.v0.Function
            codeFilename: {uploads/lambda-archive.zip}
            description: "Handler for calls to resource path : /items"
            enableCORS: true
            handler: lambda.handler
            memorySize: "128"
            name: handler-name
            runtime: nodejs6.10
            timeout: "3"
            environment:
              {MENU_TABLE_NAME}: ___DYNAMIC_PREFIX___{-bbq_menu_item}
              {ORDERS_TABLE_NAME}: ___DYNAMIC_PREFIX___{-bbq_orders}
              {RESTAURANTS_TABLE_NAME}: ___DYNAMIC_PREFIX___-{bbq_restaurants}
          "/items/{proxy+}": !com.amazonaws.mobilehub.v0.Function
            codeFilename: {uploads/lambda-archive.zip}
            description: "Handler for calls to resource path : /items/{proxy+}"
            enableCORS: true
            handler: lambda.handler
            memorySize: "128"
            name: handler-name
            runtime: nodejs6.10
            timeout: "3"
            environment:
              {MENU_TABLE_NAME}: ___DYNAMIC_PREFIX___{-bbq_menu_item}
              {ORDERS_TABLE_NAME}: ___DYNAMIC_PREFIX___{-bbq_orders}
              {RESTAURANTS_TABLE_NAME}: ___DYNAMIC_PREFIX___-{bbq_restaurants}
. . .

uploads:
  - !com.amazonaws.mobilehub.v0.Upload
    fileName: {lambda-archive.zip}
    targetS3Bucket: deployments
  - !com.amazonaws.mobilehub.v0.Upload
    fileName: {lambda.jar}
    targetS3Bucket: deployments
. . .
```

4. Save the `.yml` file and rezip the files within the uncompressed project definition file (not the folder containing those files, because that causes a path error).
5. Test your revised project export definition by importing it through the Mobile Hub console. You can verify your environmental variables through the Lambda console.

> **Note**
> By default, the Mobile Hub NoSQL Database feature configures a table's permissions to grant read and write access for Lambda functions. The kind of custom IAM policy configuration required to change the table's permissions is not included in the export of a project. An importer of a project dependent on custom policy needs enough information to recreate the policy once they have imported the project. For such a case, we recommend you provide both your policy JSON and step by step instructions (console or AWS CLI) on how and where to attach it. For more information on those steps, see Authentication and Access Control for Amazon DynamoDB.

Importing Cross-Origin Resource Sharing (CORS) Configuration

By default, AWS security infrastructure prevents calls to an API Gateway API from a browser. Configuring CORS for each path of your API securely enables your API calls over the web. CORS configuration is not included in Mobile Hub project export. The following steps describe how to manually include import of CORS configuration in your project export file.

To include CORS configuration for your |ABP| API paths

1. Unzip your exported project definition `.zip` file.

2. Open the export's `mobile-hub-project.yml` file in an editor.
3. For each API path, add a key named `enableCORS` with the value `true` under `... paths: "/items/. . ."": !com.amazonaws.mobilehub.v0.Function`, as shown in the following fragment.

```
--- !com.amazonaws.mobilehub.v0.Project
    features:
      cloudlogic: !com.amazonaws.mobilehub.v0.CloudLogic
        components:
          ReactSample: !com.amazonaws.mobilehub.v0.API
            attributes:
              name: ReactSample
              requires-signin: false
            paths:
              "/items/{proxy+}": !com.amazonaws.mobilehub.v0.Function
                name: FirstHandler
                handler: lambda.handler
                enableCORS: true
                runtime: nodejs6.10
                . . .
```

4. Rezip the files within the uncompressed project definition file (not the folder containing those files, because that causes a path error).

AWS Mobile Hub Features

AWS Mobile Hub is a service that enables even a novice to easily deploy and configure mobile app backend features using a range of powerful AWS services.

You create a free project, then choose and configure mobile app features using a point and click console. Mobile Hub takes care of the complexities in the background and then supplies you with step by step integration instructions.

Topics

Cloud Logic

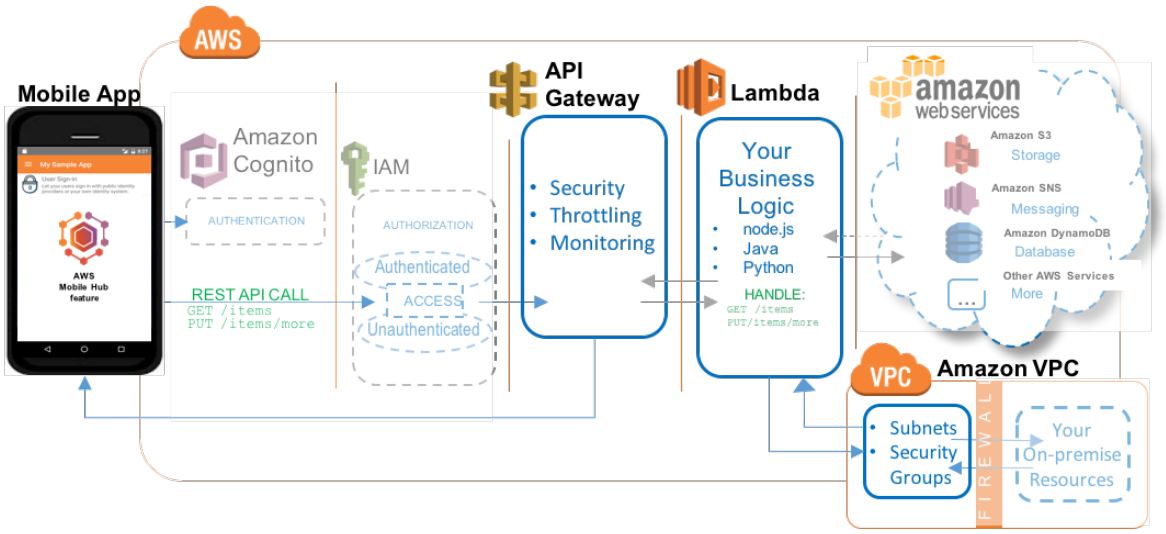
Choose the AWS Mobile Hub Cloud Logic mobile backend service feature to:

- Add business logic functions in the cloud with no cost for server set up or maintenance
- Extend your app to other services within AWS and beyond

Create a free Mobile Hub project and add the Cloud Logic feature.

Feature Details

The following image show Cloud Logic using the combination of Amazon API Gateway and AWS Lambda to implement serverless business logic and extension to other services.



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.

The Lambda functions you create are exposed to your app as a REST API by Amazon API Gateway which also provides a single secure endpoint with flexible traffic monitoring and throttling capabilities.

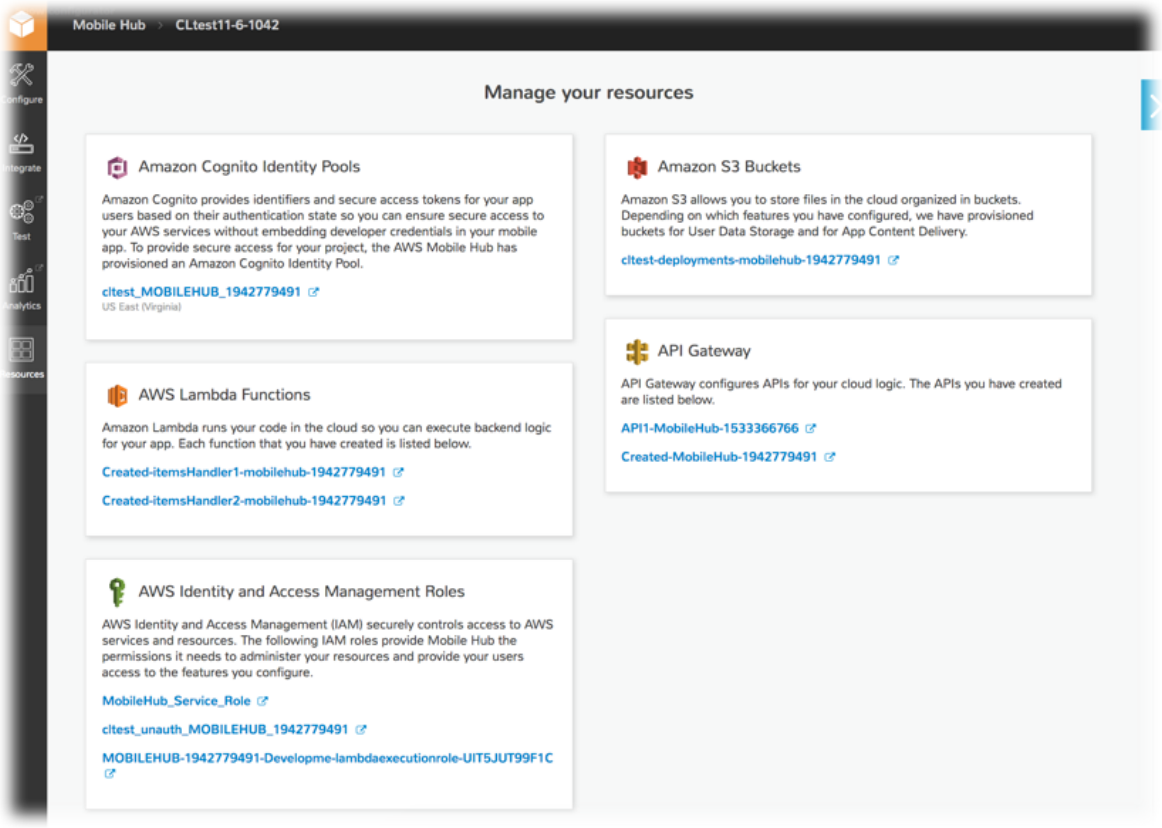
Cloud Logic At a Glance

AWS services and resources configured	• **Amazon API Gateway** (see Amazon API Gateway Developer Guide) Concepts | Console | Pricing • **AWS Lambda** (see AWS Lambda Developer Guide) Concepts | Console | Pricing • **Amazon Virtual Private Cloud** (see Amazon VPC User Guide) Concepts | Console | Pricing • **AWS CloudFormation** (see AWS CloudFormation User Guide) Concepts | Console | Pricing Mobile Hub-enabled features use Amazon Cognito for authentication and IAM for authorization. For more information, see User Sign-in (p. 348). For more information, see Viewing AWS Resources Provisioned for this Feature (p. 334).
Configuration options	This feature enables the following mobile backend capabilities:

	• Provides a default Hello World Lambda function that accepts the parameter value entered by the app user and returns it back to an app. • Enables you to choose an existing function from the list provided or use the AWS Lambda console to create new functions.
Quickstart app demos	This feature adds the following functionality to a quickstart app generated by Mobile Hub: • User can specify an AWS Lambda function by name, provide parameters and call a function and see the value returned by the function

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.

NoSQL Database

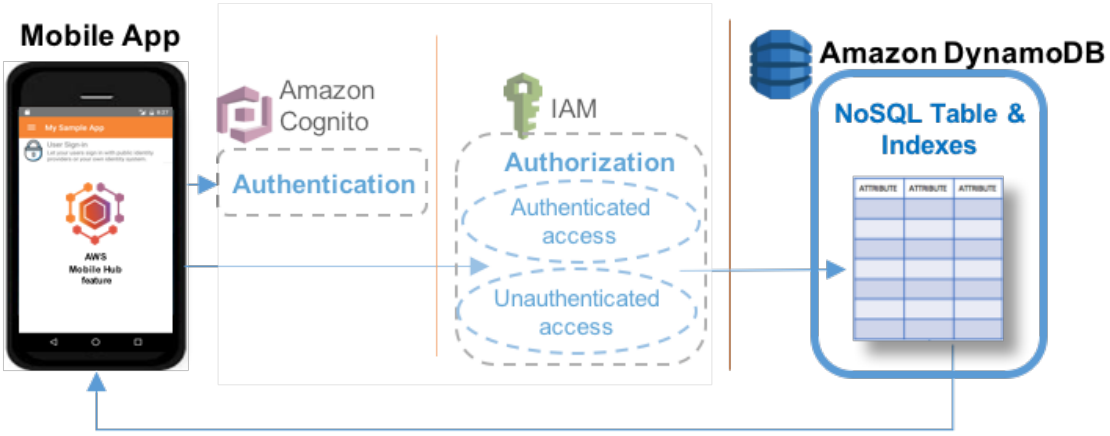
Choose the Mobile Hub NoSQL Database mobile backend feature to:

- Add easy to develop database capabilities with scalable performance and cost

Create a free Mobile Hub project and add the NoSQL DB feature in minutes.

Feature Details

The following image shows the typical connection between a mobile app and Amazon DynamoDB using the NoSQL pattern.



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*.

NoSQL Database At a Glance

AWS services and resources configured	• **Amazon DynamoDB Tables** (see Working with Tables in DynamoDB) Concepts | Console | Pricing Mobile Hub-enabled features use Amazon Cognito for authentication and IAM for authorization. For more information, see User Sign-in (p. 348). For more information, see Viewing AWS Resources Provisioned for this Feature (p. 339).
Configuration options	This feature enables the following mobile app backend capabilities: Configuring Your Tables (p. 337) - Using custom schema, based on a sample schema provided, or

	by using a wizard that guides you through choices while creating a table. Data Permissions (p. 337) - Access to your app's data can be: • **Public** (enables any mobile app user to read or write any item in the table). • **Protected** (enables any mobile app user to read any item in the table but only the owner of an item can update or delete it). • **Private** (enables only the owner of an item to read and write to a table) For more information, see Configuring the NoSQL Database Feature (p. 336). For more information, see Configuring the NoSQL Database Feature (p. 336).
Quickstart app demos	This feature adds the following to a quickstart app generated by Mobile Hub: • Insert and remove sample data, based on the schema you specify in the console. • 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.

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 |AMH| 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 |AMH| 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. 337).
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.

Topics

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

Best practice for data security is to allow the minimum access to your tables that will support your app design. Mobile Hub provides two methods to protect your data: user authentication using the User Sign-in (p. 348) feature; and NoSQL Database data table user permissions.

Note: When NoSQL Database is enabled your app communicates directly with the DynamoDB service. If you do not make the User Sign-in (p. 348) feature **Required** then, where not blocked by table user permissions, unauthenticated users will have access to read and/or write data.

Grant Permissions Only to Authenticated Users

Unless users who have not signed-in need to read or write data in a table in your app, scope down access by requiring users to sign in (authenticate) before they are allowed to use app features that perform database operations. The AWS Mobile Hub User Sign-in (p. 348) feature offers a range of methods for authenticating users that includes: federating with a sign-in provider like Facebook, Google, Active Directory, or your existing custom service. In a few clicks, you can also create your own sign-in provider backed by AWS services.

To add User Sign-in to your app, use the **Configure more features button** on a feature configuration page, or the **Configure** icon on the left. Then choose and enable **User Sign-in**.

Grant Permissions to Table Data Items Per User

When you create a new table in NoSQL Database, you choose between **Public**, **Private**, or **Protected** options, to determine which app users can read or write the table's data. Mobile Hub attaches a fine-grained access control policy to the table, that can restrict the operations available to a user based on whether or not they are the creator of data being accessed.

Public

- Public permissions allow all users to read or update all items (data rows) in the table.

Protected

- Protected permissions allow all users to read all items in the table, but only the owner of an item can update or delete that item.

Private

- Private permissions allow only the owner of an item to read or write to it.

> **Note**
> Users own a data item if their Amazon Cognito identity ID matches the value of the item's primary key.
> If you choose **Protected** or **Private** permissions for a table, then the partition key of the table must be `userId`, and be of type `string`. Secondary indexes for protected or private tables follow the same pattern as primary indexes.
> When a user creates an item in a protected or private table, AWS populates the value of the item's primary key with that user's Amazon Cognito identity ID.
> Enforcement happens when a data operation is attempted on a protected or private item. IAM will check if the item's `userId` matches the current user's Amazon Cognito identity ID, and allow or prevent the operation based on the policy attached to the table.
> When you choose **Public**, permissions for a table there is no ownership enforcement. There are no restrictions on name or data type of the primary key and secondary index primary keys of a public table.

Managing Permissions to Restricted Items 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 `orders` 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`.

Retrieving Data

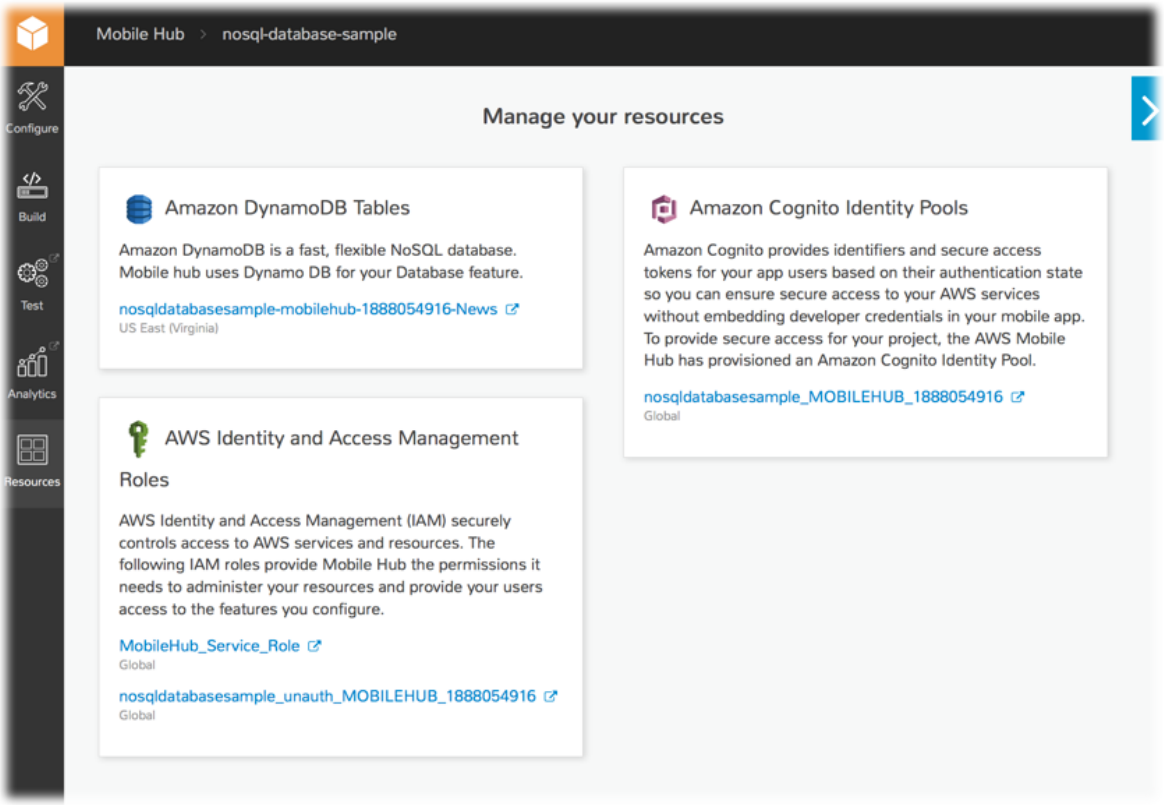
The operations you can use to retrieve data from your NoSQL database include the following:

- `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 Filter`s, which performs a :code:`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.

Messaging and Analytics

Choose the AWS Mobile Hub Messaging and Analytics feature to:

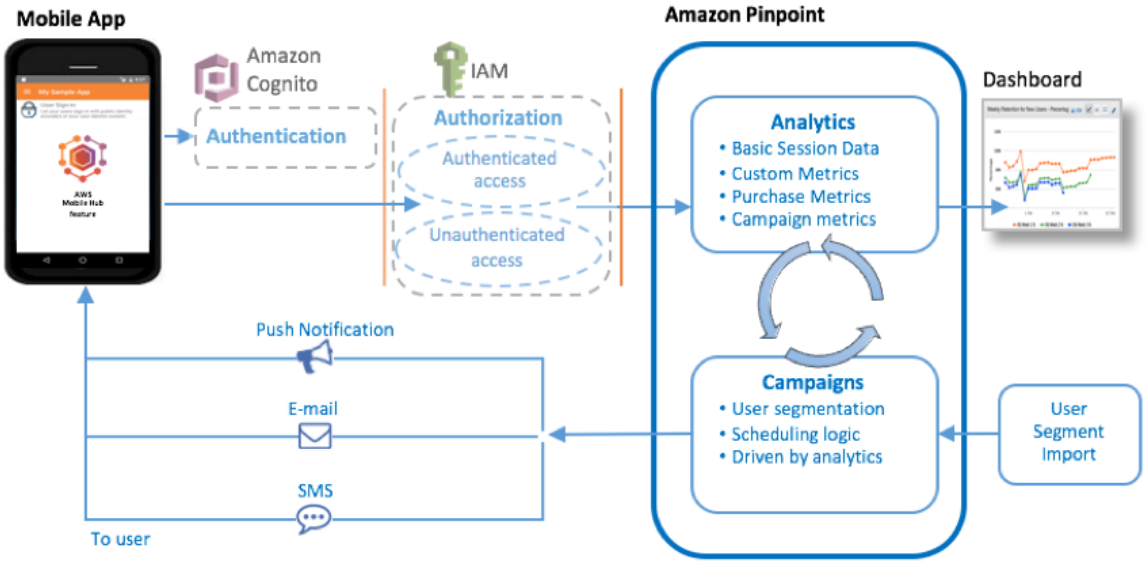
- Gather data to understand your app users' behavior
- Use that information to add campaigns to engage with your users through push notification, e-mail, and SMS

Create a free Mobile Hub project and add the Messaging and Analytics feature.

Feature Details

AWS Mobile Hub Messaging and Analytics (formerly User Engagement) helps you understand how your users use your app. It enables you to engage them through push notification, e-mail, or SMS. You can tie your analytics to your messaging so that what you communicate flows from users' behavior.

The following image shows Messaging and Analytics using Amazon Pinpoint to collect usage data from a mobile app. Amazon Pinpoint then sends messaging to selected app users based on the campaign logic designed for the app.



You can configure messaging and analytics functions separately, or use the two together to carry out campaigns to interact with your users based on the their app usage. You can configure which users receive a campaign's messaging, as well as the conditions and scheduling logic for sending messages. You can configure notifications to communicate text or cause a programatic action, such as opening an application or passing custom JSON to your client.

When you choose **Analytics**, Amazon Pinpoint performs capture, visualization, and analysis of app usage and campaign data:

- By default, Amazon Pinpoint gathers app usage session data.
- If you configure a campaign, metrics about your campaign are included.
- If you add custom analytics to your app, you can configure Amazon Pinpoint to visualize those metrics and use the data as a factor in your campaign behavior. To learn more about integrating custom analytics, see Integrating Amazon Pinpoint With Your App in the *Amazon Pinpoint User Guide*.

- Amazon Pinpoint enables you to construct funnel analytics, which visualize how many users complete each of a series of step you intend them to take in your app.
- To perform more complex analytics tasks, such as merging data from more than one app or making flexible queries, you can configure Amazon Pinpoint to stream your data to Kinesis. To learn more about using Amazon Pinpoint and Kinesis together, see Streaming Amazon Pinpoint Events to Amazon Kinesis.

When you choose **Messaging** you can configure your project to enable Amazon Pinpoint to send:

- Send Push Notifications to your Android users, through Firebase/Google Cloud Messaging, or iOS, through APNs
- E-mails to your app users using the sender ID and domain of your choice
- SMS messages

Once you have enabled Messaging and Analytics options in your Mobile Hub project, use the Amazon Pinpoint console to view visualizations of your analytics or configure your user segments and campaigns. You can also import user segment data into Amazon Pinpoint to use campaigns for any group of users.

Messaging and Analytics At a Glance

AWS services and resources configured	• **Amazon Pinpoint** (see Amazon Pinpoint Developer Guide) Concepts | Console Mobile Hub-enabled features use Amazon Cognito for authentication and IAM for authorization. For more information, see User Sign-in (p. 348).

Configuration options	This feature enables the following mobile backend capabilities:
	
	• Gather and visualize analytics of your app users' behavior.
	  • Integrate Amazon Pinpoint user engagement campaigns into your mobile app.
	  • Communicate to app users using push notifications through APNs, GCM, and FCM.
	  • via **Firebase or Google Cloud Messaging (FCM/GCM)** (see Setting Up Android Push Notifications)
	  • via **Apple Push Notification service (APNs)** (see Setting Up iOS Push Notifications)
	
	For more information, see Configuring Push Notification.
	  • Communicate to app users through e-mail.
	  • Communicate to app users through SMS.

Quickstart app demos	This feature adds **User Engagement** functionality to a quickstart app generated by Mobile Hub: • Demonstrate enabling the app user to receive campaign notifications. The app user can cause

	events that generate session, custom, campaign and purchase data. Analytics for these events is available in the Amazon Pinpoint console in close to real time. • Demonstrate providing the app user with a view of an Amazon Pinpoint data visualization, on their mobile phone.

Hosting and Streaming

Choose AWS Mobile Hub Hosting and Streaming to:

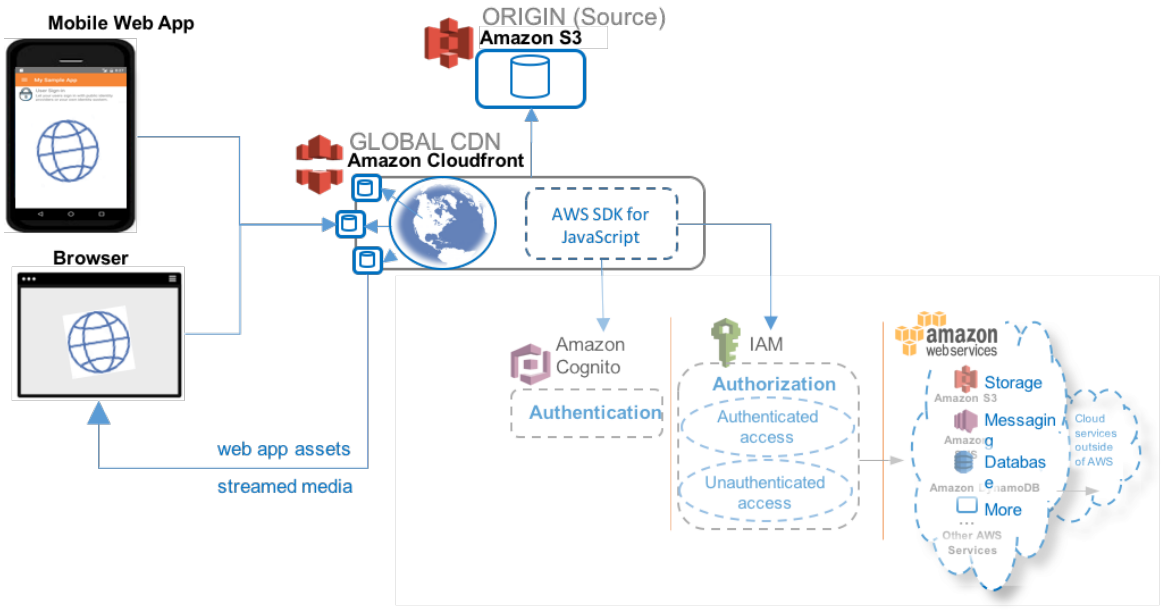
- Host content for you mobile web, native mobile or hybrid app
- Distribute your content through a global Content Delivery Network (CDN)
- Stream your media

Create a free Mobile Hub project with Hosting and Streaming. Get a custom sample app and SDK.

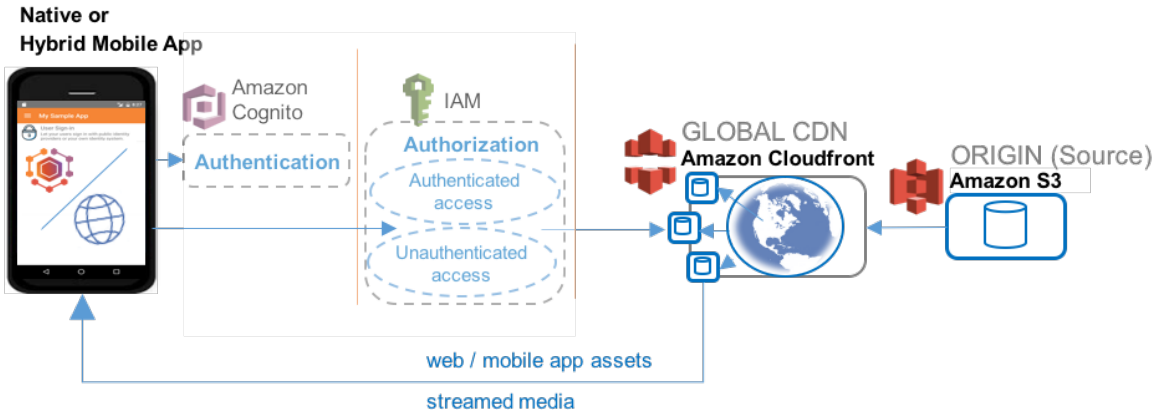
Feature Details

The Hosting and Streaming feature delivers content through a global network of endpoints using Amazon Simple Storage Service (Amazon S3) and Amazon CloudFront.

The following image shows how website assets and streaming media are distributed to a mobile web app or browser. The web app is shown requesting AWS credentials and accessing AWS services through the AWS SDK for JavaScript.



The following image shows a native or hybrid mobile app requesting AWS credentials to access content from a CDN edge location.



The Hosting and Streaming feature enables you to host website and app assets in the cloud, such as HTML, JavaScript, image, or media files. Mobile Hub creates a content source storage location (origin) using an Amazon S3 bucket. The bucket is made accessible to the internet through the Amazon S3 ability to statically host web content with no web server.

Low latency access to your content is provided to users in all regions by caching your source content on a global network of edge locations. This Content Distribution network (CDN) is provided through an Amazon CloudFront distribution which also supports media file streaming (see Amazon CloudFront streaming).

Hosting and Streaming At a Glance

AWS services and resources configured	• **Amazon CloudFront - Content Delivery Network** (see Amazon CloudFront) Concepts | Console | Pricing • **Amazon S3 Bucket** (see *Amazon S3 Getting Started Guide* <https://aws.amazon.com/cloudfront/pricing/> `__) Concepts | Console | Pricing Mobile Hub-enabled features use Amazon Cognito for authentication and IAM for authorization. For more information, see User Sign-in (p. 348). For more information, see Viewing AWS Resources Provisioned for this Feature (p. 345).
Configuration options	This feature enables the following mobile backend capabilities: • **Web app content hosting** (Internet access for your content, no web servers required) • **AWS SDK for JavaScript** (Call AWS services via standard scripting) • **Global CDN** (Global content distribution and media streaming) CloudFront offers several options for regional scope and cost of your distribution. For more information,

	see Configuring the Hosting and Streaming Feature (p. 344).
Web app demo	Sample • The AWS SDK for Javascript and a custom-generated configuration file are provisioned to your bucket. For more information, see Web App Support (p. 344).
Quickstart native app demos	This feature adds the following to a quickstart app generated by Mobile Hub: • View file list in AWS storage, download and view files, and manage their local cache.

Web App Support

When you enable Hosting and Streaming, Mobile Hub provisions a local copy of the AWS SDK for JavaScript in the root of your bucket.

Mobile Hub also generates the project configuration files `aws-config.js` and `aws-exports.js`, which contain endpoint constants for each AWS services Mobile Hub configured for your project. `aws-exports.js` is provided for integration with ES6 compatible scripting languages like Node.js. Use these values to make SDK calls to your services from your hosted web app.

> **Note**
> Best security practice is to reduce access to an app's resources as much as possible. These configuration files are publically accessible and contain identifiers for all of your app's AWS resources. If it suits your design, we recommend you protect your resources by allowing only authenticated users to access them. You can do this in this project by enabling the Mobile Hub User Sign-in (p. 348) with the **Require sign-in** option.

You can also copy the appropriate configuration file into your hybrid native/web mobile app to enable calling your AWS services from your app using JavaScript.

Configuring the Hosting and Streaming Feature

Topics

Browsing Your Content

With Hosting and Streaming enabled, you have several options:

- **Launch from Amazon S3**: This option browses to the un-cached index.html in the root of your source bucket.
- **Launch from Amazon CloudFront**: This option browses to the index.html that is cached on the CDN edge servers.

  > **Note**
  > Provisioning the edge locations for the distribution can take up to an hour. This link will not resolve until the distribution finishes propagating in the network.

- **Manage files**: This option opens the Amazon S3 console to review and manage the contents of your source bucket. You can also find your bucket in the Amazon S3 console by opening your project in Mobile Hub and then choosing the **Resources** icon on the left. The name of the bucket configured for Hosting and Streaming contains the string `hosting`.

Managing Your App Assets

You can choose from a variety of ways to manage your web app assets through use of the Amazon S3 console, the AWS Command Line Interface (CLI) or one of the many third party applications available.

Using the Amazon S3 Console

To use the Amazon S3 console to review, upload, move or delete your files stored in your bucket, navigate to the Amazon S3 console and choose the bucket whose name contains your project name. Your web app content will reside in the root folder.

Using AWS CLI

AWS CLI allows you to review, upload, move or delete your files stored in your bucket using the command line.

To install and configure the AWS CLI client, see Getting Set Up with the AWS Command Line Interface.

As an example, the sync command enables transfer of files to and from your local folder (`source`) and your bucket (`destination`).

```
$ aws s3 sync {source destination} [--options]
```

The following command syncs all files from your current local folder to the folder in your web app's bucket defined by `path`.

```
$ aws s3 sync . s3://my-web-app-bucket/path
```

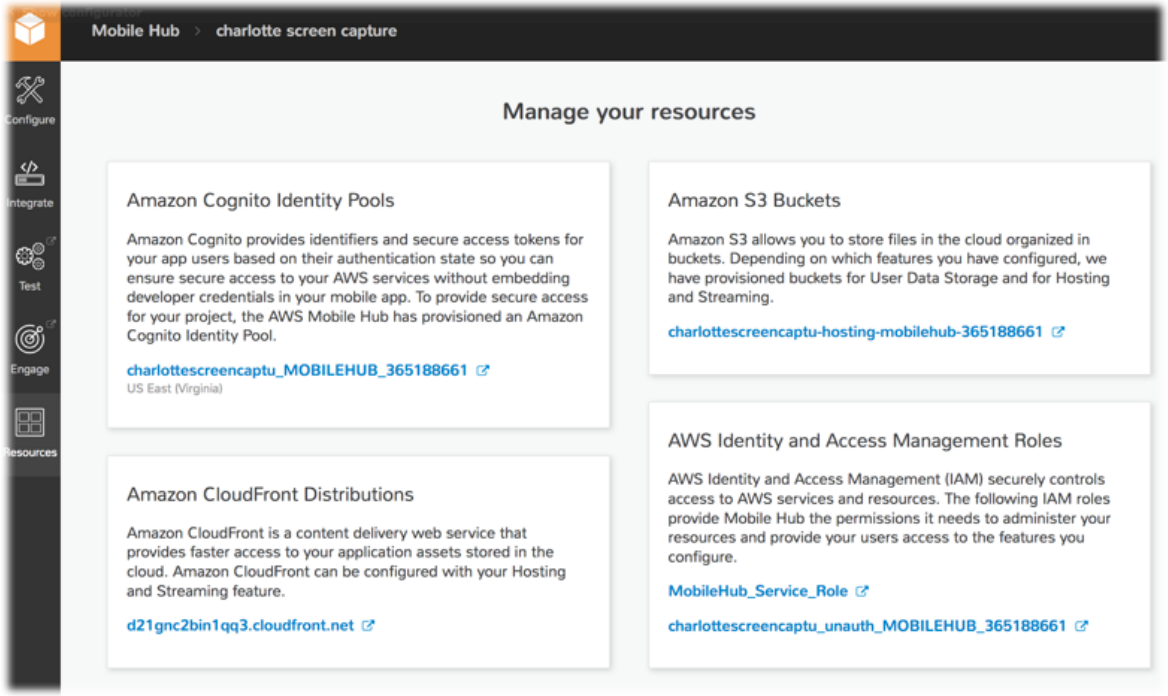
To learn more about using AWS CLI to manage Amazon S3, see Using Amazon S3 with the AWS Command Line Interface

Using a Custom Domain for Your Web App

To configure your Hosting and Streaming CDN as the destination of your custom domain, see Routing Traffic to an Amazon CloudFront Web Distribution by Using Your Domain Name.

Viewing AWS Resources Provisioned for this Feature

The following image shows the Mobile Hub**Resources** pane displaying elements typically provisioned for the Hosting and Streaming feature.



Quickstart App Details

In the Mobile Hub quickstart app, the Hosting and Streaming 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.

Conversational Bots

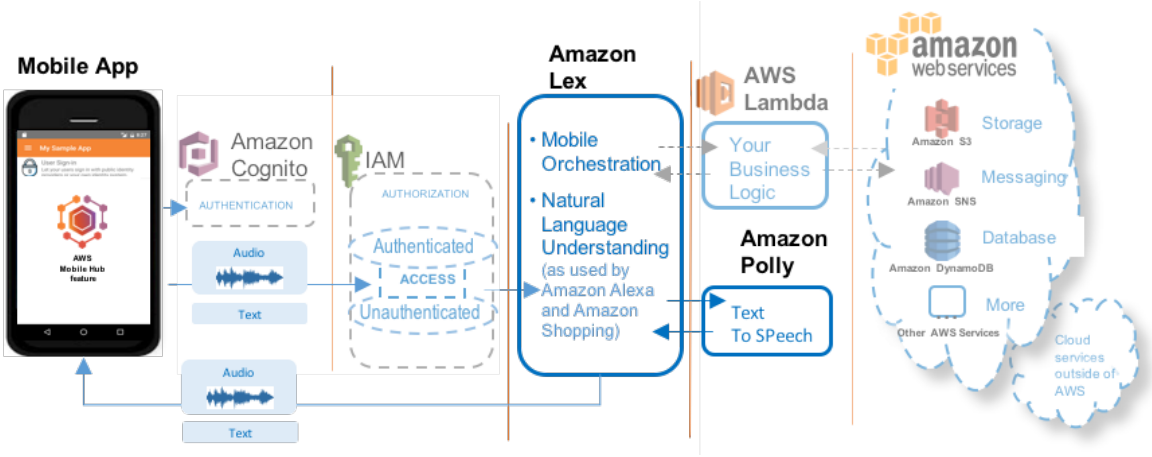
Choose the AWS Mobile Hub conversational bots mobile backend service feature to:

- Add voice and text natural language understanding interface to your app
- Use natural language voice and text to interact with your business logic in the cloud

Create a free Mobile Hub project and add the Conversational Bots feature.

Feature Details

The following image shows Conversational Bots using Amazon Lex to add natural language to a mobile app interface and as an integration point for other services.



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

AWS services and resources configured	• Amazon Lex (see Amazon Lex Developer Guide) Concepts | Console Mobile Hub-enabled features use Amazon Cognito for authentication and IAM for authorization. For more information, see User Sign-in (p. 348).
Configuration options	This feature enables the following mobile backend capabilities: • 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. • 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.
Quickstart app demos	This feature adds the following functionality to a quickstart app generated by Mobile Hub:

	• Enables user to interact with a conversational bot that interacts with Amazon Lex.

User Sign-in

Choose the AWS Mobile Hub User Sign-in mobile backend feature to:

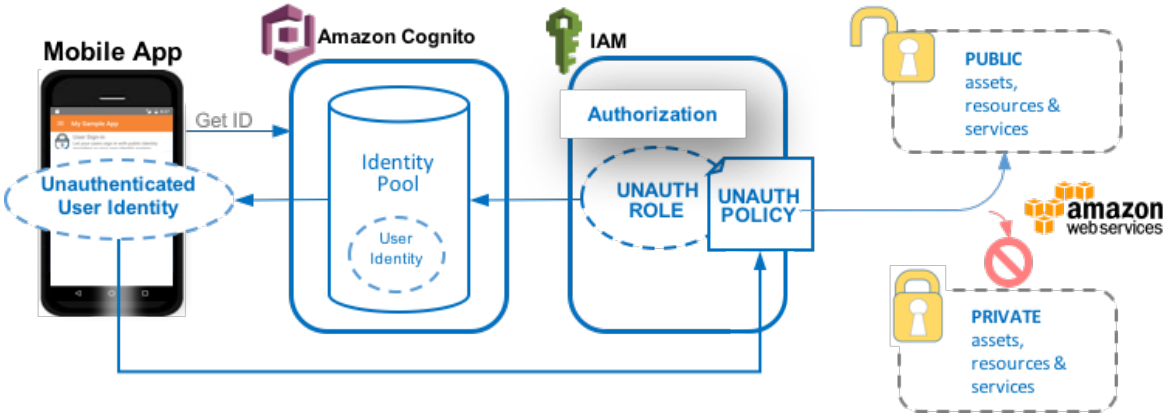
- Add AWS user authentication and secure identity access management to your mobile app.

  Note: Secure unauthenticated access to AWS resources is available to all Mobile Hub projects with or without the User Sign-in feature.
- Enable your users to sign-in to access AWS resources with existing credentials from identity providers like Facebook, Google, Microsoft Active Directory Federation Services or your own custom user directory.

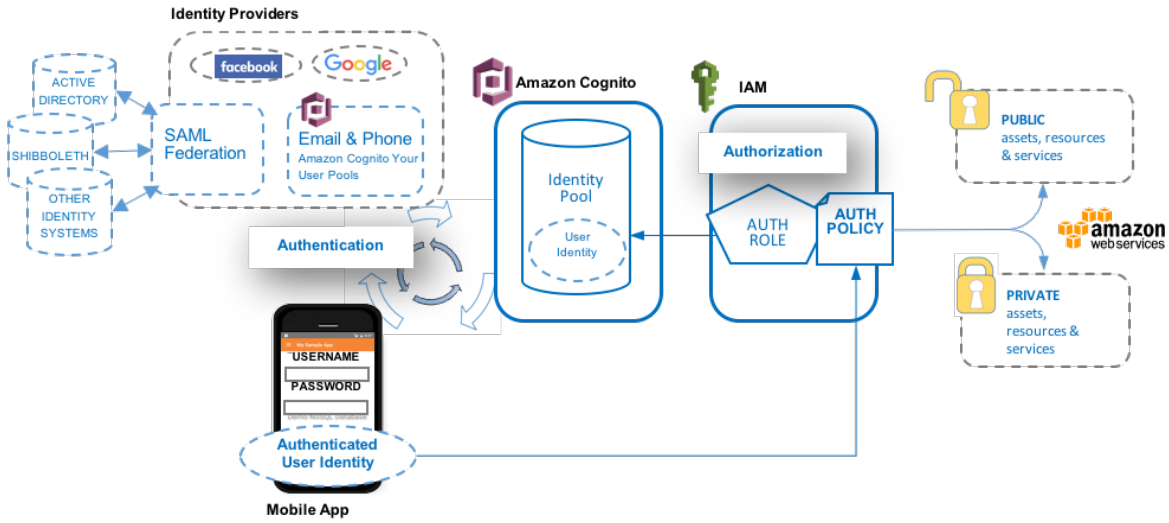
Create a free Mobile Hub project and add the User Sign-in feature.

Feature Details

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.

User Sign-in Feature At a Glance

AWS services and resources configured	• **Amazon Cognito** Concepts | Console | Pricing • **Amazon Cognito Identity Pool** (see Using Federated Identities) • **Amazon Cognito Your User Pools** (see Creating and Managing User Pools) • **Amazon Cognito SAML Federation** (see Overview of Configuring SAML 2.0-Based Federation) • **IAM role and security policies** (see Control Access to Mobile Hub Projects (p. 314)) Concepts | Console | Pricing For more information, see Viewing AWS Resources Provisioned for this Feature (p. 352).
Configuration options	This feature enables the following mobile backend capabilities: **Sign-in Providers** (users gain greater access when they sign in) • via **Google** authentication (see Set Up Your Backend (p. 20)) • via **Facebook** authentication (see Set Up Your Backend (p. 20)) • via **Email and Password** authentication (see User Sign-in Providers (p. 350)) • via **SAML Federation** authentication (see User Sign-in Providers (p. 350))

	Required Sign-in (authenticated access) **Optional Sign-in** (users gain greater access when they sign in) For more information, see Configuring User Sign-in (p. 350)
Quickstart demo features	This feature adds the following to a quickstart app generated by Mobile Hub: • Unauthenticated access (if allowed by your app's configuration), displaying the ID that AWS assigns to the app instance's device. • Sign-in screen that authenticates users using the selected method: Facebook, Google, or Email and Password (your own user pool). • With **Optional Sign-in** and **Require Sign-in**, the app demonstrates an access barrier to protected folders for unauthenticated users.

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. 20).

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. 161).

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:

  1. View your IdP's documentation to understand how to use the AWS federation metadata file to register AWS as a service provider.
  2. Ensure that your Mobile Hub project is configured to use Email & Password sign-in to create an Amazon Cognito User Pool.
  3. Configure your IdP as an Identity Provider for your user pool using the steps on Creating SAML Identity Providers for Your User Pool.

  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.

Note
If user sign-in is not required, unauthenticated users can access to data in your database tables and files in your storage buckets, unless those resources are explicitly restricted through another mechanism.

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

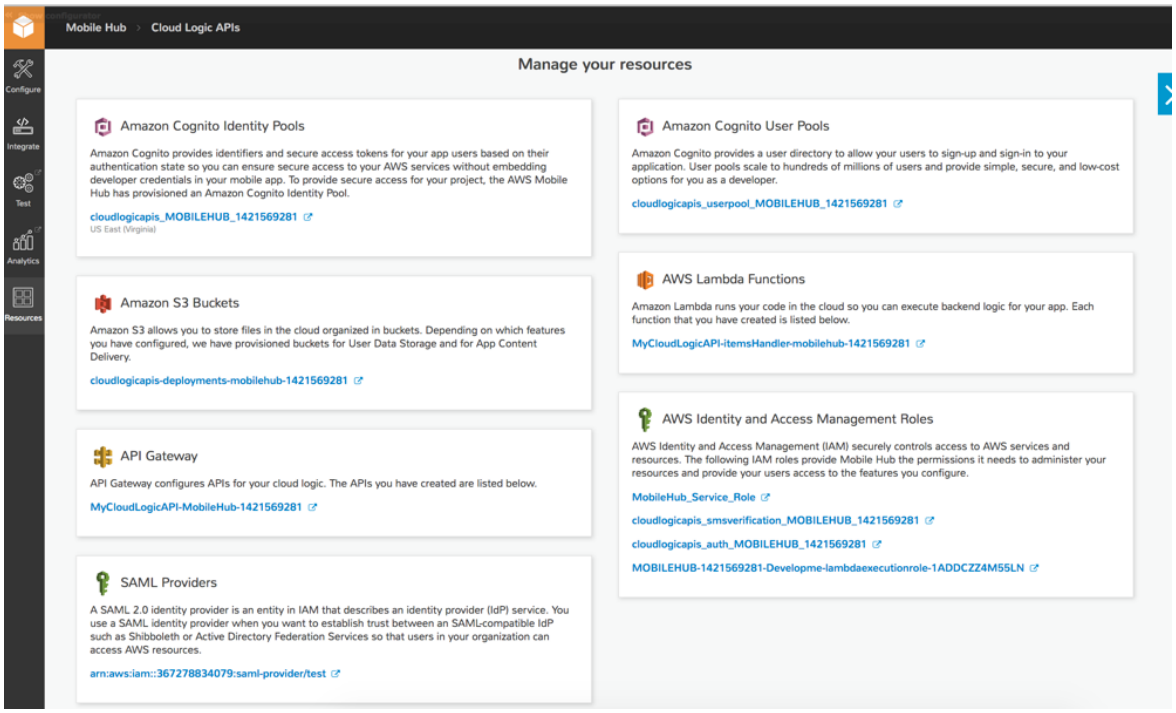
When your mobile app is saved, Mobile Hub creates an Amazon Cognito identity pool and a new IAM role. These are used to generate temporary AWS credentials for the quickstart app users to access your AWS resources. The AWS IAM role security policies are updated based on the sign-in features enabled.

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 Control Access to Mobile Hub Projects (p. 314).

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.

User File Storage

Choose AWS Mobile Hub User File Storage to:

- Add cloud storage of user files, profile data, and app state to your mobile app
- Use fine-grained control of access to files and data, implementing four common patterns of permissions policy

Looking for Amazon Cognito Sync?	Amazon Cognito Sync has been deprecated. For real time data sync between devices, with built-in offline capabilities, see AWS AppSync.

Create a free Mobile Hub project and add the User File Storage feature.

Feature Details

The Mobile Hub User File Storage feature, creates and configures four folders for each user, inside an Amazon Simple Storage Service (Amazon S3) bucket belonging to the app.

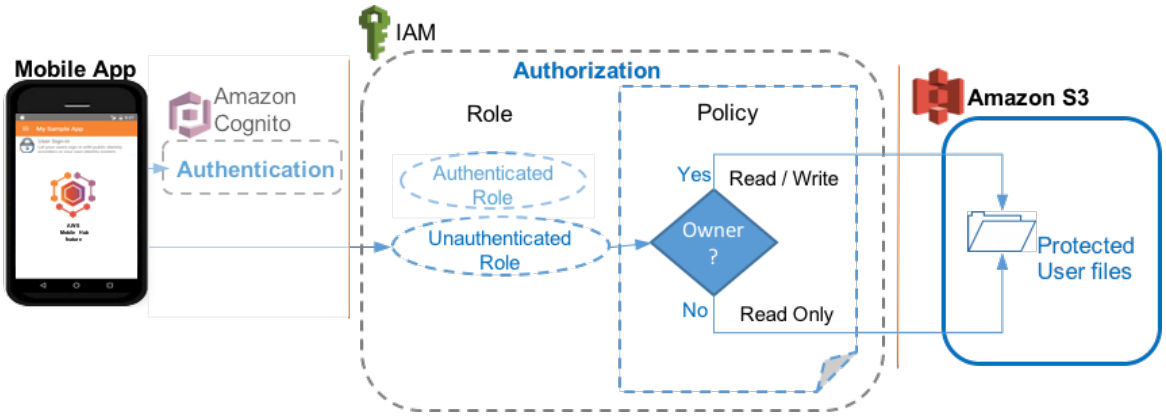
Best practice for app security is to allow the minimum access to your buckets that will support your app design. Each of the four folders provisioned has a policy illustrating different permissions choices attached. In addition, Mobile Hub provides the option to restrict access to your app to only authenticated users using the User Sign-in (p. 348) feature.

Note: If you do not make the User Sign-in (p. 348) feature **Required** then, where not blocked by a folder or bucket access policy, unauthenticated users will have access to read and/or write data.

The following table shows the details of permissions policies that are provisioned for each folder type.

Folder name	Owner permissions	Everyone else permissions
Public	Read/Write	Read/Write
Private	Read/Write	None
Protected	Read/Write	Read Only
Uploads	Write Only	Write Only

The following image shows IAM policy being applied to control file access in a Protected folder. The policy grants read/write permissions for the user who created the folder, and read only permissions for everyone else.



The User File 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.

User File Storage At a Glance

AWS services and resources configured	• **Amazon S3 bucket** (see Amazon S3 Getting Started Guide) Concepts | Console | Pricing Mobile Hub-enabled features use Amazon Cognito for authentication and IAM for authorization. For more information, see User Sign-in (p. 348). For more information, see Viewing AWS Resources Provisioned for this Feature (p. 355).

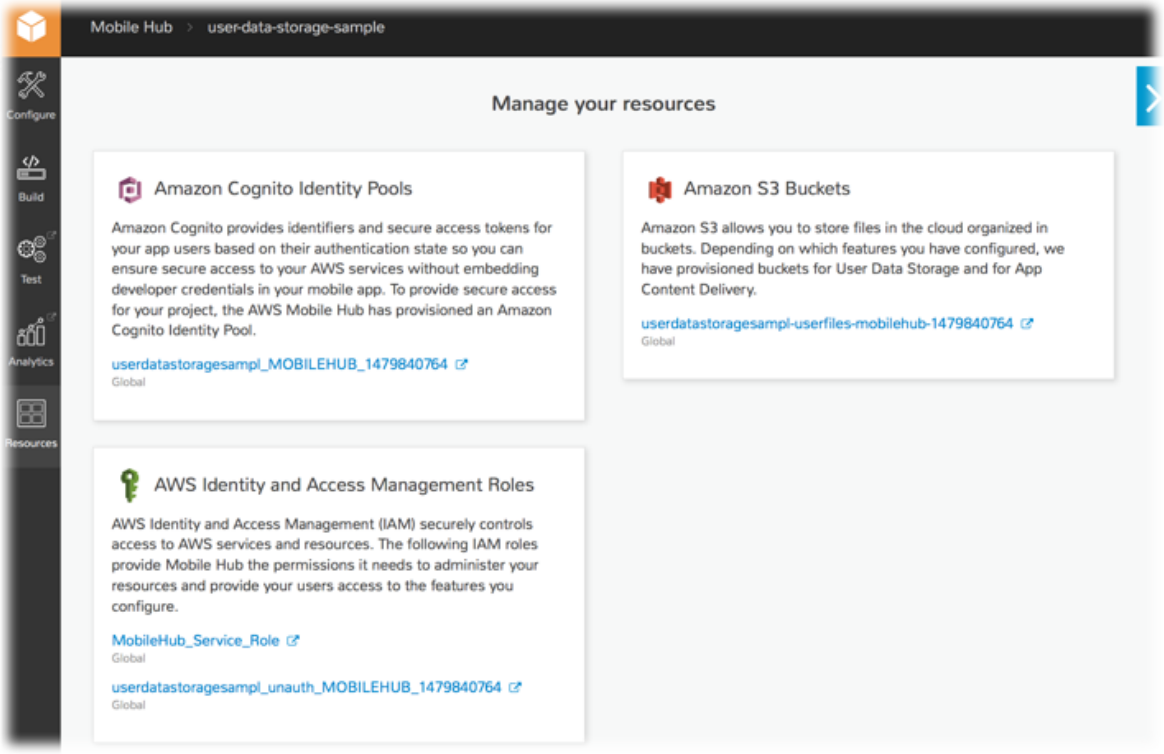
Configuration options	This feature enables the following configuration options mobile backend capabilities:
	
	• Store user files and app data using Amazon S3. When you enable User File Storage four folders are provisioned, each with a distinct access policy configuration:
	  • `private` - Each mobile app user can create, read, update, and delete their own files in this folder. No other app users can access this folder.
	  • `protected` - 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.
	  • `public` ? Any app user can create, read, update, and delete files in this folder.

Quickstart demo features	This feature adds the following to a quickstart app generated by Mobile Hub: • File explorer for the app's S3 bucket allows the user to:

	• Upload and view files in any **Public** folder. • View and download files in a **Private** folder that the user created. • View and download files in a **Protected** folder anyone created and upload files to that folder if the user created it. • Upload files to any **Uploads** folder. User setting of choice of color theme can be persisted to and retrieves from the cloud.

Viewing AWS Resources Provisioned for this Feature

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



Mobile Hub Project Service Region Hosting

The configuration settings of your Mobile Hub project are stored in the AWS US East (Virginia) region.

The AWS services you configure are hosted in the region you select for your project, if they are available in that region. If services are not available in that region, then Mobile hub will host the services in another region.

For more details about regional endpoints, see AWS Regions and Endpoints.

To understand where services for your project will be hosted, find the region for your project in the following tables.

Select your project's region:

- US East (Virginia) (p. 356)

US East (Virginia)

Hosting for these services:	Is located in:
Amazon API Gateway (Cloud Logic)	US East (Virginia)
Amazon Cognito (User Sign-in / User File Storage)	US East (Virginia)
Amazon DynamoDB (NoSQL Database)	US East (Virginia)
Amazon Lex (Conversational Bots)	US East (Virginia)
Amazon Pinpoint (Messaging and Analytics)	US East (Virginia)
Amazon S3 (User File Storage / Messaging and Hosting)	US East (Virginia)
AWS Lambda (Cloud Logic)	US East (Virginia)

US East (Ohio)

Hosting for these services:	Is located in:
Amazon API Gateway (Cloud Logic)	US East (Ohio)
Amazon Cognito (User Sign-in / User File Storage)	US East (Ohio)
Amazon DynamoDB (NoSQL Database)	US East (Ohio)
Amazon Lex (Conversational Bots)	US East (Virginia)
Amazon Pinpoint (Messaging and Analytics)	US East (Virginia)
Amazon S3 (User File Storage / Messaging and Hosting)	US East (Ohio)
AWS Lambda (Cloud Logic)	US East (Ohio)

US West (California)

Hosting for these services:	Is located in:
Amazon API Gateway (Cloud Logic)	US West (California)
Amazon Cognito (User Sign-in / User File Storage)	US West (Oregon)
Amazon DynamoDB (NoSQL Database)	US West (California)
Amazon Lex (Conversational Bots)	US East (Virginia)
Amazon Pinpoint (Messaging and Analytics)	US East (Virginia)
Amazon S3 (User File Storage / Messaging and Hosting)	US West (California)
AWS Lambda (Cloud Logic)	US West (California)

US West (Oregon)

Hosting for these services:	Is located in:
Amazon API Gateway (Cloud Logic)	US West (Oregon)
Amazon Cognito (User Sign-in / User File Storage)	US West (Oregon)
Amazon DynamoDB (NoSQL Database)	US West (Oregon)
Amazon Lex (Conversational Bots)	US East (Virginia)
Amazon Pinpoint (Messaging and Analytics)	US East (Virginia)
Amazon S3 (User File Storage / Messaging and Hosting)	US West (Oregon)
AWS Lambda (Cloud Logic)	US West (Oregon)

EU West (Ireland)

Hosting for these services:	Is located in:
Amazon API Gateway (Cloud Logic)	EU West (Ireland)
Amazon Cognito (User Sign-in / User File Storage)	EU West (Ireland)
Amazon DynamoDB (NoSQL Database)	EU West (Ireland)
Amazon Lex (Conversational Bots)	US East (Virginia)
Amazon Pinpoint (Messaging and Analytics)	US East (Virginia)
Amazon S3 (User File Storage / Messaging and Hosting)	EU West (Ireland)

Hosting for these services:	Is located in:
AWS Lambda (Cloud Logic)	EU West (Ireland)

EU West (London)

Hosting for these services:	Is located in:
Amazon API Gateway (Cloud Logic)	EU West (London)
Amazon Cognito (User Sign-in / User File Storage)	EU West (London)
Amazon DynamoDB (NoSQL Database)	EU West (London)
Amazon Lex (Conversational Bots)	US East (Virginia)
Amazon Pinpoint (Messaging and Analytics)	US East (Virginia)
Amazon S3 (User File Storage / Messaging and Hosting)	EU West (London)
AWS Lambda (Cloud Logic)	EU West (London)

EU (Frankfurt)

Hosting for these services:	Is located in:
Amazon API Gateway (Cloud Logic)	EU (Frankfurt)
Amazon Cognito (User Sign-in / User File Storage)	EU (Frankfurt)
Amazon DynamoDB (NoSQL Database)	EU (Frankfurt)
Amazon Lex (Conversational Bots)	US East (Virginia)
Amazon Pinpoint (Messaging and Analytics)	US East (Virginia)
Amazon S3 (User File Storage / Messaging and Hosting)	EU (Frankfurt)
AWS Lambda (Cloud Logic)	EU (Frankfurt)

Asia Pacific (Tokyo)

Hosting for these services:	Is located in:
Amazon API Gateway (Cloud Logic)	Asia Pacific (Tokyo)
Amazon Cognito (User Sign-in / User File Storage)	Asia Pacific (Tokyo)
Amazon DynamoDB (NoSQL Database)	Asia Pacific (Tokyo)

Hosting for these services:	Is located in:
Amazon Lex (Conversational Bots)	US East (Virginia)
Amazon Pinpoint (Messaging and Analytics)	US East (Virginia)
Amazon S3 (User File Storage / Messaging and Hosting)	Asia Pacific (Tokyo)
AWS Lambda (Cloud Logic)	Asia Pacific (Tokyo)

Asia Pacific (Seoul)

Hosting for these services:	Is located in:
Amazon API Gateway (Cloud Logic)	Asia Pacific (Seoul)
Amazon Cognito (User Sign-in / User File Storage)	Asia Pacific (Seoul)
Amazon DynamoDB (NoSQL Database)	Asia Pacific (Seoul)
Amazon Lex (Conversational Bots)	US East (Virginia)
Amazon Pinpoint (Messaging and Analytics)	US East (Virginia)
Amazon S3 (User File Storage / Messaging and Hosting)	Asia Pacific (Seoul)
AWS Lambda (Cloud Logic)	Asia Pacific (Seoul)

Asia Pacific (Mumbai)

Hosting for these services:	Is located in:
Amazon API Gateway (Cloud Logic)	Asia Pacific (Mumbai)
Amazon Cognito (User Sign-in / User File Storage)	Asia Pacific (Mumbai)
Amazon DynamoDB (NoSQL Database)	Asia Pacific (Mumbai)
Amazon Lex (Conversational Bots)	US East (Virginia)
Amazon Pinpoint (Messaging and Analytics)	US East (Virginia)
Amazon S3 (User File Storage / Messaging and Hosting)	Asia Pacific (Mumbai)
AWS Lambda (Cloud Logic)	Asia Pacific (Mumbai)

Asia Pacific (Singapore)

Hosting for these services:	Is located in:
Amazon API Gateway (Cloud Logic)	Asia Pacific (Singapore)
Amazon Cognito (User Sign-in / User File Storage)	Asia Pacific (Singapore)
Amazon DynamoDB (NoSQL Database)	Asia Pacific (Singapore)
Amazon Lex (Conversational Bots)	US East (Virginia)
Amazon Pinpoint (Messaging and Analytics)	US East (Virginia)
Amazon S3 (User File Storage / Messaging and Hosting)	Asia Pacific (Singapore)
AWS Lambda (Cloud Logic)	Asia Pacific (Singapore)

Asia Pacific (Sydney)

Hosting for these services:	Is located in:
Amazon API Gateway (Cloud Logic)	Asia Pacific (Sydney)
Amazon Cognito (User Sign-in / User File Storage)	Asia Pacific (Sydney)
Amazon DynamoDB (NoSQL Database)	Asia Pacific (Sydney)
Amazon Lex (Conversational Bots)	US East (Virginia)
Amazon Pinpoint (Messaging and Analytics)	US East (Virginia)
Amazon S3 (User File Storage / Messaging and Hosting)	Asia Pacific (Sydney)
AWS Lambda (Cloud Logic)	Asia Pacific (Sydney)

South America (São Paulo)

Hosting for these services:	Is located in:
Amazon API Gateway (Cloud Logic)	South America (São Paulo)
Amazon Cognito (User Sign-in / User File Storage)	US East (Virginia)
Amazon DynamoDB (NoSQL Database)	South America (São Paulo)
Amazon Lex (Conversational Bots)	US East (Virginia)
Amazon Pinpoint (Messaging and Analytics)	US East (Virginia)
Amazon S3 (User File Storage / Messaging and Hosting)	US East (Virginia)

Hosting for these services:	Is located in:
AWS Lambda (Cloud Logic)	South America (São Paulo)

Mobile Hub Project Troubleshooting

The following sections describe issues you might encounter when setting up, importing or exporting Mobile Hub projects, and their remedies.

Topics

Cannot Import an API

Error Message

- `Project owner does not own existing API : arn:aws:execute-api:us-east-1:012345678901:abcdefghij.`

  (where the API identifier arn:aws:execute-api:us-east-1:012345678901:abcdefghij is specific to the project being imported)

Description

- This message means that the API with the ID shown cannot be imported because it does not exist in the current AWS account. This occurs when the APIs in the original project were created outside of the Mobile Hub Cloud Logic feature and then imported.

Remedy

- **To remedy this condition, take the following steps.**
  1. Modify the YAML of the project definition you are importing by removing the sections under the `features:components` node that begin with the name of an API that was imported into the original project's Cloud Logic feature.
  2. Save and import the project definition.
  3. Enable the Mobile Hub Cloud Logic feature in your imported project and recreate the API and its handler.

Cannot Import a NoSQL Table

Error Message

- There is already an existing DynamoDB table called 'someprojectname-mobilehub-012345678-TableName' in your account. Please choose a different name or remove the existing table and retry your request.

  (where the table name someprojectname-mobilehub-012345678-TableName is specific to the project being imported)

Description

- This message occurs when you import a project containing the NoSQL Database Feature. It indicates that the Amazon DynamoDB table in the project configuration already exists. This can occur when a YAML tablename value was edited in the project definition file and there is more than one attempt to import it into the same account.

Remedy

- **To remedy this condition, take the following steps**
  1. Modify any tablename values to remove the conflict.
  2. Save and import the project definition.
  3. Adjust the code of the imported app where it references the old tablename value.

Cannot Import Multiple NoSQL Tables

Error Message

- Project file(s) cannot be decoded. They may contain data that was encrypted by a different account. Failed to decode push feature. Failed to decode credential attribute.

Description

- This message occurs when you import Push Notifications messaging service credentials or Amazon SNS topic identifiers for features that are not associated with your AWS account.

Remedy

- **To remedy this condition, take the following steps**
  1. Modify the YAML of the project definition you are importing by removing table definition sections.
  2. Save and import the project definition.
  3. Use the table definitions you removed to manually create those tables using the Mobile Hub NoSQL Database feature.

Cannot Import Push Credentials

Error Message

- Project file(s) cannot be decoded. They may contain data that was encrypted by a different account. Failed to decode push feature. Failed to decode credential attribute.

Description

- This message occurs when you import Push Notifications messaging service credentials or Amazon SNS topic identifiers for features that are not associated with your AWS account.

Remedy

- **To remedy this condition, take the following steps**
  1. Modify the YAML of the project definition you are importing by removing the push: node.
  2. Save and import the project definition.
  3. Enable the Mobile Hub Push Notifications or User Engagement feature using your own messaging service credentials and topics.

Build Artifacts Can't be Found

Error Message

- Unable to find build artifact uploads/exported-project-definition.zip in Amazon S3 bucket archive-deployments-mobilehub-0123456789 for project-name.

  where exported-project-definition, the numerical portion of the Amazon S3 bucket identifier, and the project-name are specific to the project being imported)

Description

- This message occurs when a project import fails because Mobile Hub can't find the file of a Cloud Logic API handler function (Lambda) that is specified in the .yml project definition file.

Remedy

- **To remedy this condition, take the following steps**

  The remedy for this condition is to make the location of the Lambda file(s) match the path specified in the project definition YAML.

  The error occurs if, for any reason, the path described in the codeFilename: key in the YAML does not match the actual location of the Lambda function file relative to the root of the `...-deployments-...` Amazon S3 bucket that Mobile Hub deploys when Cloud Logic is enabled. For more information, see Importing API Handlers for Cloud Logic APIs (p. 329).

```
--- !com.amazonaws.mobilehub.v0.Project
features:
  cloudlogic: !com.amazonaws.mobilehub.v0.CloudLogic
    components:
      api-name: !com.amazonaws.mobilehub.v0.API
        attributes:
          name: api-name
          requires-signin: true
          sdk-generation-stage-name: Development
        paths:
          /items: !com.amazonaws.mobilehub.v0.Function
            codeFilename: uploads/lambda-archive.zip
            description: "Handler for calls to resource path : /items"
            enableCORS: true
            handler: lambda.handler
            memorySize: "128"
            name: handler-name
            runtime: nodejs6.10
            timeout: "3"
          "/items/{proxy+}": !com.amazonaws.mobilehub.v0.Function
            codeFilename: uploads/lambda-archive.zip
            description: "Handler for calls to resource path : /items/{proxy+}"
            enableCORS: true
```

```
            handler: lambda.handler
            memorySize: "128"
            name: handler-name
            runtime: nodejs6.10
            timeout: "3"
. . .
```

Potential reasons include:

- A typo in the path value of the `uploads: fileName` key in the YAML.
- A path error caused during manual modifications to a project definition .zip file.

  To make a project's Cloud Logic API handler Lambda functions available for import, an author must unzip, modify, and rezip the exported project file. If the uncompressed project definition file folder is rezipped, rather than zipping the contents within that folder, the path is changed so that the original archive is inside of a new archive folder. If the path to an original export named `your-project.zip` was `lambda-archive.zip`, then the path would change to `your-project/lambda-archive.zip`. You can remedy this by modifying the uploads: fileName value or rezipping the project export file contents without the including the folder.
- A missing Lambda file in a project definition file containing a YAML file that specifies a path for `uploads: fileName`.

Unable to Configure S3 Bucket During

Error Message

- It looks like there was a problem creating or configuring your S3 bucket.

Description

- Mobile Hub was unable to create a S3 bucket for your project's deployment artifacts during Mobile Hub project import.

Remedy

- **To remedy this condition, try the following steps**

  Check that you are not at maximum bucket capacity using the Amazon S3 console.

Administrator Required Error During Setup

Error Message

- It looks like you do not have permission for this operation.

Description

- The user does not have permission to create the required Mobile Hub Service Role during configuration of a Mobile Hub project.

Remedy

- **To remedy this condition, try the following steps**

Contact an administrator for your AWS account and ask them to create the service role at the following location: https://console.aws.amazon.com/mobilehub/home#/activaterole/.

Account Setup Incomplete

Error Message

- It looks like your AWS account is not fully set up.

Description

- This error can occur for a range of reasons during Mobile Hub project configuration.

Remedy

- **To remedy this condition, try the following steps**
  - Sign out of the AWS console and lose down all browser windows. Then try to log in to the <problematic>`AWS Mobile console <>`__</problematic>
    and attempt the operation that initially caused the er.
  - If the issue persists, post to the
    <problematic>`AWS Mobile Development forum<https://forums.aws.amazon.com/forum.jspa?forumID=88>`__</problematic>
    for support.

File Too Large to Import

Error Message

- The project file is too large. The max file size is 10 MB.

Description

- This message occurs when you attempt to import a project definition file that is larger than 10MB.

Remedy

- Reduce the size of the project export file. Project exporters may want to deliver large file payloads outside of their project definition files, along with providing instructions for importers about how to use AWS consoles to incorporate those accompanying files.

AWS Amplify Library for Web

AWS Amplify is an open source JavaScript library for frontend and mobile developers building cloud-enabled applications. The library is a declarative interface across different categories of operations in order to make common tasks easier to add into your application. The default implementation works with Amazon Web Services (AWS) resources but is designed to be open and pluggable for usage with other cloud services that wish to provide an implementation or custom backends.

The AWS Mobile CLI, built on AWS Mobile Hub, provides a command line interface for frontend JavaScript developers to seamlessly enable and configure AWS services into their apps. With minimal configuration, you can start using all of the functionality provided by the AWS Mobile Hub from your favorite terminal application.

Topics

Get Started

Overview

The AWS Mobile CLI provides a command line experience that allows frontend JavaScript developers to quickly create and integrate AWS backend resources into their mobile apps.

Prerequisites

1. Sign up for the AWS Free Tier.
2. Install Node.js with NPM.
3. Install AWS Mobile CLI

```
npm install -g awsmobile-cli
```

4. Configure the CLI with your AWS credentials

   To setup permissions for the toolchain used by the CLI, run:

```
awsmobile configure
```

   If prompted for credentials, follow the steps provided by the CLI. For more information, see provide IAM credentials to AWS Mobile CLI (p. 395).

Set Up Your Backend

Need to create a quick sample React app? See Create a React App.

To configure backend features for your app

1. In the root folder of your app, run:

```
awsmobile init
```

   The `init` command creates a backend project for your app. By default, analytics and web hosting are enabled in your backend and this configuration is automatically pulled into your app when you initialize.

2. When prompted, provide the source directory for your project. The CLI will generate `aws-exports.js` in this location. This file contains the configuration and endpoint metadata used to link your frontend to your backend services.

```
? Where is your project's source directory:  src
```

3. Respond to further prompts with the following values.

```
? Where is your project's distribution directory to store build artifacts:  build
? What is your project's build command:  npm run-script build
? What is your project's start command for local test run:  npm run-script start
? What awsmobile project name would you like to use:  YOUR-APP-NAME-2017-11-10-15-17-48
```

After the project is created you will get a success message which also includes details on the path where the aws-exports.js is copied.

```
awsmobile project's details logged at: awsmobilejs/#current-backend-info/backend-
details.json
awsmobile project's access information logged at: awsmobilejs/#current-backend-info/aws-
exports.js
awsmobile project's access information copied to: src/aws-exports.js
awsmobile project's specifications logged at: awsmobilejs/#current-backend-info/mobile-hub-
project.yml
contents in #current-backend-info/ is synchronized with the latest information in the aws
 cloud
```

Your project is now initialized.

Connect to Your Backend

AWS Mobile uses the open source AWS Amplify library to link your code to the AWS features configured for your app.

This section of the guide shows examples using a React application of the kind output by `create-react-app` or a similar tool.

To connect the app to your configured AWS features

In `index.js` (or in other code that runs at launch-time), add the following imports.

```
import Amplify from 'aws-amplify';
import awsmobile from './YOUR-PATH-TO/aws-exports';
```

Then add the following code.

```
Amplify.configure(awsmobile);
```

Run Your App Locally

Your app is now ready to launch and use the default features configured by AWS Mobile.

To launch your app locally in a browser

In the root folder of your app, run:

```
awsmobile run
```

Behind the scenes, this command runs `npm install` to install the Amplify library and also pushes any backend configuration changes to AWS Mobile. To run your app locally without pushing backend changes you cou can choose to run `npm install` and then run `npm start`.

Anytime you launch your app, app analytics are gathered and can be visualized (p. 370) in an AWS console.

AWS Free Tier	Initializing your app or adding features through the CLI will cause AWS services to be configured on your behalf. The pricing for AWS Mobile services enables you to learn and prototype at little or no cost using the AWS Free Tier.

Next Steps

Topics

Deploy your app to the cloud

Using a simple command, you can publish your app's frontend to hosting on a robust content distribution network (CDN) and view it in a browser.

To deploy your app to the cloud and launch it in a browser

In the root folder of your app, run:

```
awsmobile publish
```

To push any backend configuration changes to AWS and view content locally, run `awsmobile run`. In both cases, any pending changes you made to your backend configuration are made to your backend resources.

By default, the CLI configures AWS Mobile Hosting and Streaming (p. 342) feature, that hosts your app on Amazon CloudFront CDN endpoints. These locations make your app highly available to the public on the Internet and support media file streaming

You can also use a custom domain (p. 383) for your hosting location.

Test Your App on Our Mobile Devices

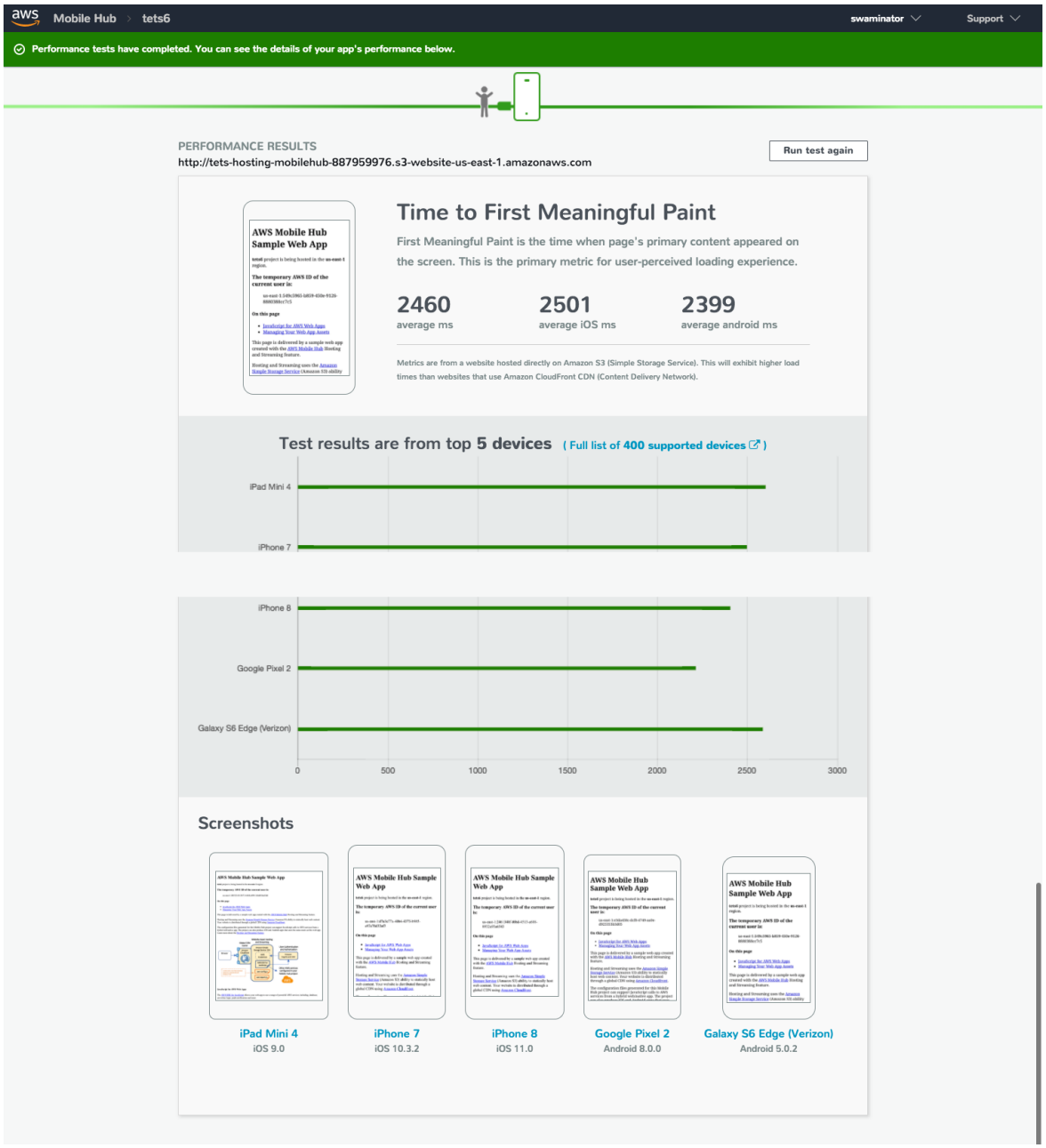
Invoke a free remote test of your app on a variety of real devices and see results, including screen shots.

To invoke a remote test of your app

In the root folder of your app, run:

```
awsmobile publish --test
```

The CLI will open the reporting page for your app in the Mobile Hub console to show the metrics gathered from the test devices. The device that runs the remote test you invoke resides in AWS Device Farm which provides flexible configuration of tests and reporting.



Add Features

Add the following AWS Mobile features to your mobile app using the CLI.

- Analytics (p. 370)
- User Sign-in (p. 371)
- NoSQL Database (p. 372)
- User File Storage (p. 377)
- Cloud Logic (p. 379)

Learn more

To learn more about the commands and usage of the AWS Mobile CLI, see the AWS Mobile CLI reference (p. 386).

Learn about AWS Mobile Amplify.

Add Analytics

BEFORE YOU BEGIN	The steps on this page assume you have already completed the steps on Get Started (p. 366).

Basic Analytics Backend is Enabled for Your App

When you complete the AWS Mobile CLI setup and launch your app, anonymized session and device demographics data flows to the AWS analytics backend.

To send basic app usage analytics to AWS

Launch your app locally by running:

```
npm start
```

When you use your app the Amazon Pinpoint service gathers and visualizes analytics data.

To view the analytics using the Amazon Pinpoint console

1. Run `npm start`, `awsmobile run`, or `awsmobile publish --test` at least once.
2. Open your project in the AWS Mobile Hub console.

```
awsmobile console
```

3. Choose the **Analytics** icon on the left, to navigate to your project in the Amazon Pinpoint console.
4. Choose **Analytics** on the left.

You should see an up-tick in several graphs.

Add Custom Analytics to Your App

You can configure your app so that Amazon Pinpoint gathers data for custom events that you register within the flow of your code.

To instrument custom analytics in your app

In the file containing the event you want to track, add the following import:

```
import { Analytics } from 'aws-amplify';
```

Add the a call like the following to the spot in your JavaScript where the tracked event should be fired:

```
componentDidMount() {
   Analytics.record('FIRST-EVENT-NAME');
}
```

Or to relevant page elements:

```
handleClick = () => {
     Analytics.record('SECOND-EVENT-NAME');
}

<button onClick={this.handleClick}>Call request</button>
```

To test:

1. Save the changes and run `npm start`, `awsmobile run`, or `awsmobile publish --test` to launch your app. Use your app so that tracked events are triggered.
2. In the Amazon Pinpoint console, choose **Events** near the top.
3. Select an event in the **Event** dropdown menu on the left.

Custom event data may take a few minutes to become visible in the console.

Next Steps

Learn more about the analytics in AWS Mobile which are part of the Messaging and Analytics (p. 340) feature. This feature uses Amazon Pinpoint.

Learn about AWS Mobile CLI (p. 386).

Learn about AWS Mobile Amplify.

Add Auth / User Sign-in

BEFORE YOU BEGIN	The steps on this page assume you have already completed the steps on Get Started (p. 366).

Set Up Your Backend

The AWS Mobile CLI components for user authentication include a rich, configurable UI for sign-up and sign-in.

To enable the Auth features

In the root folder of your app, run:

```
awsmobile user-signin enable

awsmobile push
```

Connect to Your Backend

The AWS Mobile CLI enables you to integrate ready-made sign-up/sign-in/sign-out UI from the command line.

To add user auth UI to your app

1. Install AWS Amplify for React library.

```
npm install --save aws-amplify-react
```

2. Add the following import in `App.js` (or other file that runs upon app startup):

```
import { withAuthenticator } from 'aws-amplify-react';
```

3. Then change `export default App;` to the following.

```
export default withAuthenticator(App);
```

To test, run `npm start`, `awsmobile run`, or `awsmobile publish --test`.

Next Steps

Learn more about the AWS Mobile User Sign-in (p. 348) feature, which uses Amazon Cognito.

Learn about AWS Mobile CLI (p. 386).

Learn about AWS Mobile Amplify.

Access Your Database

BEFORE YOU BEGIN	The steps on this page assume you have already completed the steps on Get Started (p. 397).

The AWS Mobile CLI and Amplify library make it easy to perform create, read, update, and delete ("CRUD") actions against data stored in the cloud through simple API calls in your JavaScript app.

Set Up Your Backend

To create a database

1. Enable the NoSQL database feature and configure your table.

   In the root folder of your app, run:

```
awsmobile database enable --prompt
```

2. Choose `Open` to make the data in this table viewable by all users of your application.

```
? Should the data of this table be open or restricted by user?
# Open
  Restricted
```

3. For this example type in `todos` as your `Table name`.

```
? Table name: todos
```

Add columns and queries

You are creating a table in a NoSQL database and adding an initial set of columns, each of which has a name and a data type. NoSQL lets you add a column any time you store data that contains a new column. NoSQL tables must have one column defined as the Primary Key, which is a unique identifier for each row.

1. For this example, follow the prompts to add three columns: `team` (string), `todoId` (number), and `text` (string).

```
? What would you like to name this column: team
? Choose the data type: string
```

2. When prompted to `? Add another column`, type `Y` and then choose enter. Repeat the steps to create `todoId` and `text` columns.
3. Select `team` as the primary key.

```
? Select primary key
# team
  todoId
  text
```

4. Choose `(todoId)` as the sort key and then `no` to adding any more indexes, to keep the example simple.

Sort Keys and Indexes	To optimize preformance, you can define a column as a Sort Key. Choose a column to be a Sort Key if it will be frequently used in combination with the Primary key to query your table. You can also create Secondary Indexes to make addtional columns sort keys.

```
? Select sort key
# todoId
  text
  (No Sort Key)

? Add index (Y/n): n
Table todos saved.
```

The `todos` table is now created.

Use a cloud API to do CRUD operations

To access your NoSQL database, you will create an API that can be called from your app to perform CRUD operations.

Why an API?	Using an API to access your database provides a simple coding interface on the frontend and

	robust flexibility on the backend. Behind the scenes, a call to an Amazon API Gateway API end point in the cloud is handled by a serverless Lambda function.

To create a CRUD API

1. Enable and configure the Cloud Logic feature**

```
awsmobile cloud-api enable --prompt
```

2. Choose `Create CRUD API for an existing Amazon DynamoDB table` API for an exisitng Amazon DynamoDB table" and then choose enter.

```
? Select from one of the choices below. (Use arrow keys)
  Create a new API
# Create CRUD API for an existing Amazon DynamoDB table
```

3. Select the `todos` table created in the previous steps, and choose enter.

```
? Select Amazon DynamoDB table to connect to a CRUD API
# todos
```

4. Push your configuration to the cloud. Without this step, the configuration for your database and API is now in place only on your local machine.

```
awsmobile push
```

   The required DynamoDB tables, API Gateway endpoints, and Lambda functions will now be created.

Create your first Todo

The AWS Mobile CLI enables you to test your API from the command line.

Run the following command to create your first todo.

```
awsmobile cloud-api invoke todosCRUD POST /todos '{"body": {"team": "React", "todoId": 1,
 "text": "Learn more Amplify"}}'
```

Connect to Your Backend

The examples in this section show how you would integrate AWS Amplify library calls using React (see the AWS Amplify documentation to use other flavors of Javascript).

The following component is a simple Todo list that you might add to a `create-react-app` project. The Todos component currently adds and displays `todos` to and from an in memory array.

```
// To Do app example

import React from 'react';

class Todos extends React.Component {
  state = { team: "React", todos: [] };
```

```
  render() {
    let todoItems = this.state.todos.map(({todoId, text}) => {
      return <li key={todoId}>{text}</li>;
    });

    return (
      <div style={styles}>
        <h1>{this.state.team} Todos</h1>
        <ul>
          {todoItems}
        </ul>

        <form>
          <input ref="newTodo" type="text" placeholder="What do you want to do?" />
          <input type="submit" value="Save" />
        </form>
      </div>
    );
  }
}

let styles = {
  margin: "0 auto",
  width: "25%"
};

export default Todos;
```

Displaying todos from the cloud

The `API` module from AWS Amplify allows you connect to DynamoDB through API Gateway endpoints.

To retrieve and display items in a database

1. Import the `API` module from `aws-amplify` at the top of the Todos component file.

```
import { API } from 'aws-amplify';
```

2. Add the following `componentDidMount` to the `Todos` component to fetch all of the `todos`.

```
async componentDidMount() {
  let todos = await API.get('todosCRUD', `/todos/${this.state.team}`);
  this.setState({ todos });
}
```

When the `Todos` component mounts it will fetch all of the `todos` stored in your database and display them.

Saving todos to the cloud

The following fragment shows the `saveTodo` function for the Todo app.

```
async saveTodo(event) {
  event.preventDefault();

  const { team, todos } = this.state;
  const todoId = todos.length + 1;
  const text = this.refs.newTodo.value;
```

```
  const newTodo = {team, todoId, text};
  await API.post('todosCRUD', '/todos', { body: newTodo });
  todos.push(newTodo);
  this.refs.newTodo.value = '';
  this.setState({ todos, team });
}
```

Update the `form` element in the component's render function to invoke the `saveTodo` function when the form is submitted.

```
<form onSubmit={this.saveTodo.bind(this)}>
```

Your entire component should look like the following:

```
// To Do app example

import React from 'react';
import { API } from 'aws-amplify';

class Todos extends React.Component {
  state = { team: "React", todos: [] };

  async componentDidMount() {
    const todos = await API.get('todosCRUD', `/todos/${this.state.team}`)
    this.setState({ todos });
  }

  async saveTodo(event) {
    event.preventDefault();

    const { team, todos } = this.state;
    const todoId = todos.length + 1;
    const text = this.refs.newTodo.value;

    const newTodo = {team, todoId, text};
    await API.post('todosCRUD', '/todos', { body: newTodo });
    todos.push(newTodo);
    this.refs.newTodo.value = '';
    this.setState({ todos, team });
  }

  render() {
    let todoItems = this.state.todos.map(({todoId, text}) => {
      return <li key={todoId}>{text}</li>;
    });

    return (
      <div style={styles}>
        <h1>{this.state.team} Todos</h1>
        <ul>
          {todoItems}
        </ul>

        <form onSubmit={this.saveTodo.bind(this)}>
          <input ref="newTodo" type="text" placeholder="What do you want to do?" />
          <input type="submit" value="Save" />
        </form>
      </div>
    );
  }
}
```

```
let styles = {
  margin: "0 auto",
  width: "25%"
}

export default Todos;
```

Next Steps

- Learn how to retrieve specific items and more with the API module in AWS Amplify.
- Learn how to enable more features for your app with the AWS Mobile CLI.

Add Storage

BEFORE YOU BEGIN	The steps on this page assume you have already completed the steps on Get Started (p. 366).

The AWS Mobile CLI and AWS Amplify library make it easy to store and manage files in the cloud from your JavaScript app.

Set Up the Backend

Enable the User File Storage feature by running the following commands in the root folder of your app.

```
awsmobile user-files enable

awsmobile push
```

Connect to the Backend

The examples in this section show how you would integrate AWS Amplify library calls using React (see the AWS Amplify documentation to use other flavors of Javascript).

The following simple component could be added to a `create-react-app` project to present an interface that uploads images and download them for display.

```
// Image upload and download for display example component
// src/ImageViewer.js

import React, { Component } from 'react';

class ImageViewer extends Component {
  render() {
    return (
      <div>
        <p>Pick a file</p>
        <input type="file" />
      </div>
    );
  }
}

export default ImageViewer;
```

Upload a file

The `Storage` module enables you to upload files to the cloud. All uploaded files are publicly viewable by default.

Import the `Storage` module in your component file.

```
// ./src/ImageViewer.js

import { Storage } from 'aws-amplify';
```

Add the following function to use the `put` function on the `Storage` module to upload the file to the cloud, and set your component's state to the name of the file.

```
uploadFile(event) {
  const file = event.target.files[0];
  const name = file.name;

  Storage.put(key, file).then(() => {
    this.setState({ file: name });
  });
}
```

Place a call to the `uploadFile` function in the `input` element of the component's render function, to start upload when a user selects a file.

```
render() {
  return (
    <div>
      <p>Pick a file</p>
      <input type="file" onChange={this.uploadFile.bind(this)} />
    </div>
  );
}
```

Display an image

To display an image, this example shows the use of the `S3Image` component of the AWS Amplify for React library.

1. From a terminal, run the following command in the root folder of your app.

   ```
   npm install --save aws-amplify-react
   ```

2. Import the `S3Image` module in your component.

   ```
   import { S3Image } from 'aws-amplify-react';
   ```

Use the S3Image component in the render function. Update your render function to look like the following:

```
render() {
  return (
     <div>
       <p>Pick a file</p>
       <input type="file" onChange={this.handleUpload.bind(this)} />
       { this.state && <S3Image path={this.state.path} /> }
     </div>
```

```
  );
}
```

Put together, the entire component should look like this:

```
// Image upload and download for display example component

import React, { Component } from 'react';
import { Storage } from 'aws-amplify';
import { S3Image } from 'aws-amplify-react';

class ImageViewer extends Component {

  handleUpload(event) {
    const file = event.target.files[0];
    const path = file.name;
    Storage.put(path, file).then(() => this.setState({ path }) );
  }

  render() {
    return (
      <div>
        <p>Pick a file</p>
        <input type="file" onChange={this.handleUpload.bind(this)} />
        { this.state && <S3Image path={this.state.path} /> }
      </div>
    );
  }
}

export default ImageViewer;
```

Next Steps

- Learn how to do private file storage and more with the Storage module in AWS Amplify.
- Learn how to enable more features for your app with the AWS Mobile CLI.
- Learn how to use those features in your app with the AWS Amplify library.
- Learn more about the analytics for the User File Storage feature.
- Learn more about how your files are stored on Amazon Simple Storage Service.

Access Your APIs

BEFORE YOU BEGIN	The steps on this page assume you have already completed the steps on Get Started (p. 366).

The AWS Mobile CLI and Amplify library make it easy to create and call cloud APIs and their handler logic from your JavaScript.

Set Up Your Backend

Create Your API

In the following examples you will create an API that is part of a cloud-enabled number guessing app. The CLI will create a serverless handler for the API behind the scenes.

To enable and configure an API

1. In the root folder of your app, run:

```
awsmobile cloud-api enable --prompt
```

2. When prompted, name the API `Guesses`.

```
? API name: Guesses
```

3. Name a HTTP path `/number`. This maps to a method call in the API handler.

```
? HTTP path name (/items): /number
```

4. Name your Lambda API handler function `guesses`.

```
? Lambda function name (This will be created if it does not already exists): guesses
```

5. When prompted to add another HTTP path, type `N`.

```
? Add another HTTP path (y/N): N
```

6. The configuration for your Guesses API is now saved locally. Push your configuration to the cloud.

```
awsmobile push
```

To test your API and handler

From the command line, run:

```
awsmobile cloud-api invoke Guesses GET /number
```

The Cloud Logic API endpoint for the `Guesses` API is now created.

Customize Your API Handler Logic

The AWS Mobile CLI has generated a Lambda function to handle calls to the `Guesses` API. It is saved locally in `YOUR-APP-ROOT-FOLDER/awsmobilejs/backend/cloud-api/guesses`. The `app.js` file in that directory contains the definitions and functional code for all of the paths that are handled for your API.

To customize your API handler

1. Find the handler for POST requests on the `/number` path. That line starts with `app.post('number',`. Replace the callback function's body with the following:

```
# awsmobilejs/backend/cloud-api/guesses/app.js
app.post('/number', function(req, res) {
  const correct = 12;
  let guess = req.body.guess
  let result = ""

  if (guess === correct) {
    result = "correct";
  } else if (guess > correct) {
    result = "high";
```

```
  } else if (guess < correct) {
    result = "low";
  }

  res.json({ result })
});
```

2. Push your changes to the cloud.

```
awsmobile push
```

The `Guesses` API handler logic that implements your new number guessing functionality is now deployed to the cloud.

Connect to Your Backend

The examples in this section show how you would integrate AWS Amplify library calls using React (see the AWS Amplify documentation to use other flavors of Javascript).

The following simple component could be added to a `create-react-app` project to present the number guessing game.

```
// Number guessing game app example

# src/GuessNumber.js

class GuessNumber extends React.Component {
  state = { answer: null };

  render() {
    let prompt = ""
    const answer = this.state.answer

    switch (answer) {
      case "lower":
        prompt = "Incorrect. Guess a lower number."
      case "higher":
        prompt = "Incorrect. Guess a higher number."
      case "correct":
        prompt = `Correct! The number is ${this.refs.guess.value}!`
      default:
        prompt = "Guess a number between 1 and 100."
    }

    return (
      <div style={styles}>
        <h1>Guess The Number</h1>
        <p>{ prompt }</p>

        <input ref="guess" type="text" />
        <button type="submit">Guess</button>
      </div>
    )

  }
}

let styles = {
  margin: "0 auto",
  width: "30%"
```

```
};

export default GuessNumber;
```

Make a Guess

The `API` module from AWS Amplify allows you to send requests to your Cloud Logic APIs right from your JavaScript application.

To make a RESTful API call

1. Import the `API` module from `aws-amplify` in the `GuessNumber` component file.

```
import { API } from 'aws-amplify';
```

2. Add the `makeGuess` function. This function uses the `API` module's `post` function to submit a guess to the Cloud Logic API.

```
async makeGuess() {
  const guess = parseInt(this.refs.guess.value);
  const body = { guess }
  const { result } = await API.post('Guesses', '/number', { body });
  this.setState({
    guess: result
  });
}
```

3. Change the Guess button in the component's `render` function to invoke the `makeGuess` function when it is chosen.

```
<button type="submit" onClick={this.makeGuess.bind(this)}>Guess</button>
```

Open your app locally and test out guessing the number by running `awsmobile run`.

Your entire component should look like the following:

```
// Number guessing game app example

import React from 'react';
import { API } from 'aws-amplify';

class GuessNumber extends React.Component {
  state = { guess: null };

  async makeGuess() {
    const guess = parseInt(this.refs.guess.value, 10);
    const body = { guess }
    const { result } = await API.post('Guesses', '/number', { body });
    this.setState({
      guess: result
    });
  }

  render() {
    let prompt = ""

    switch (this.state.guess) {
```

```
      case "high":
        prompt = "Incorrect. Guess a lower number.";
        break;
      case "low":
        prompt = "Incorrect. Guess a higher number.";
        break;
      case "correct":
        prompt = `Correct! The number is ${this.refs.guess.value}!`;
        break;
      default:
        prompt = "Guess a number between 1 and 100.";
    }

    return (
      <div style={styles}>
        <h1>Guess The Number</h1>
        <p>{ prompt }</p>

        <input ref="guess" type="text" />
        <button type="submit" onClick={this.makeGuess.bind(this)}>Guess</button>
      </div>
    )

  }
}

let styles = {
  margin: "0 auto",
  width: "30%"
};

export default GuessNumber;
```

Next Steps

- Learn how to retrieve specific items and more with the API module in AWS Amplify.
- Learn how to enable more features for your app with the AWS Mobile CLI.
- Learn more about what happens behind the scenes, see Set up Lambda and API Gateway.

Host Your Web App

Topics

About Hosting and Streaming

The first time that you push your web app to the cloud, the Hosting and Streaming feature is enabled to statically host your app on the web. Using the AWS Mobile CLI, this happens when you first run:

```
$ awsmobile publish
```

A container for your content is created using an Amazon S3 bucket. The content is available publicly on the Internet and you can preview the content directly using a testing URL.

Content placed in your bucket is automatically distributed to a global content delivery network (CDN). Amazon CloudFront implements the CDN which can host your app on an endpoint close to every user, globally. These endpoints can also stream media content. To learn more, see CloudFront Streaming Tutorials.

By default, Hosting and Streaming deploys a simple sample web app that accesses AWS services.

Managing Your App Assets

You can use the AWS Mobile CLI or the Amazon S3 console to manage the content of your bucket.

Use the AWS CLI to Manage Your Bucket Contents

AWS CLI allows you to review, upload, move or delete your files stored in your bucket using the command line. To install and configure the AWS CLI client, see Getting Set Up with the AWS Command Line Interface.

As an example, the sync command enables transfer of files to and from your local folder (`source`) and your bucket (`destination`).

```
$ aws s3 sync {source destination} [--options]
```

The following command syncs all files from your current local folder to the folder in your web app's bucket defined by `path`.

```
$ aws s3 sync . s3://my-web-app-bucket/path
```

To learn more about using AWS CLI to manage Amazon S3, see Using Amazon S3 with the AWS Command Line Interface

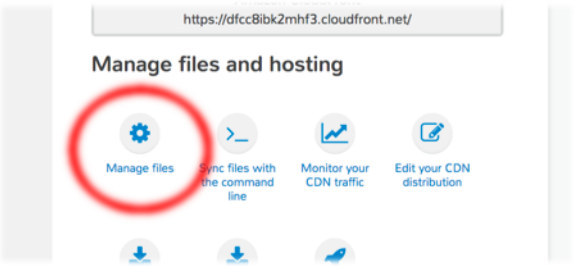
Use the Amazon S3 Console to Manage Your Bucket

To use the Amazon S3 console to review, upload, move or delete your files stored in your bucket, use the following steps.

1. From the root of your project, run:

   ```
   awsmobile console
   ```

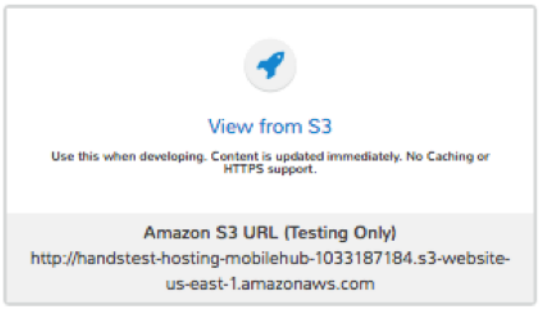
2. Choose the tile with the name of your project, then choose the Hosting and Streaming tile.
3. Choose the link labelled **Manage files** to display the contents of your bucket in the Amazon S3 console.



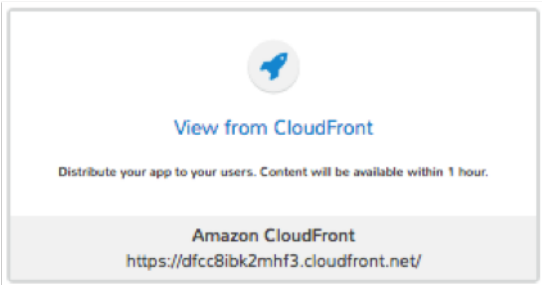
Other Useful Functions in the AWS Mobile Hub Console

The Mobile Hub console also provides convenient ways to browse to your web content, return to the AWS CLI content on this page, and other relevant tasks. These include:

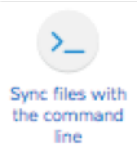
- The **View from S3** link browses to the web contents of your bucket. When Hosting and Streaming is enabled, the bucket is populated with the files for a default web app files that is viewable immediately.



- The **View from CloudFront** browses to the web contents that have propagated from your bucket to CDN. The endpoint propagation is dependent on network conditions. You can expect your content to be distributed and viewable within one hour.



- The **Sync files with the command line** link takes you to content on this page that describes how to use the command line to manage the web app and streaming media files in your bucket.



Configure a Custom Domain for Your Web App

To use your custom domain for linking to your Web app, use the Route 53 service to configure DNS routing.

For a web app hosted in a single location, see Routing Traffic to a Website that Is Hosted in an Amazon S3 Bucket.

For a web app distributed through a global CDN, see Routing Traffic to an Amazon CloudFront Web Distribution by Using Your Domain Name

AWS Mobile Hub Features

The following pages contain reference material for the AWS Mobile CLI for Web (JavaScript).

Topics

AWS Mobile CLI Reference

The AWS Mobile CLI provides a command line interface for frontend JavaScript developers to seamlessly enable AWS services and configure into their apps. With minimal configuration, you can start using all of the functionality provided by the the AWS Mobile Hub from your favorite terminal program.

Installation and Usage

This section details the usage and the core commands of the `awsmobile CLI` for JavaScript.

Install AWS Mobile CLI

1. Sign up for the AWS Free Tier.
2. Install Node.js with NPM.
3. Install AWS Mobile CLI

```
npm install -g awsmobile-cli
```

4. Configure the CLI with your AWS credentials

   To setup permissions for the toolchain used by the CLI, run:

   ```
   awsmobile configure
   ```

   If prompted for credentials, follow the steps provided by the CLI. For more information, see provide IAM credentials to AWS Mobile CLI (p. 395).

Usage

The AWS Mobile CLI usage is designed to resemble other industry standard command line interfaces.

```
awsmobile <command> [options]
```

The `help` and `version` options are universal to all the commands. Additional special options for some commands are detailed in the relevant sections.

```
-V, --version  output the version number
-h, --help     output usage information
```

For example:

```
awsmobile -help
or
awsmobile init --help
```

Summary of CLI Commands

The current set of commands supported by the `awsmobile CLI` are listed below.

awsmobile init (p. 387)	Initializes a new Mobile Hub project, checks for IAM keys, and pulls the aws-exports.js file
awsmobile configure (p. 388)	Shows existing keys and allows them to be changed if already set. If keys aren't set, deep

	links the user to the IAM console to create keys and then prompts for the access key and secret key. This command helps edit configuration settings for the aws account or the project.
awsmobile pull (p. 389)	Downloads the latest aws-exports.js, YAML or any other relevant project details from the Mobile Hub project
awsmobile push (p. 389)	Uploads local metadata, Lambda code, Dynamo definitions or any other relevant project details to Mobile Hub
awsmobile publish (p. 389)	Executes `awsmobile push`, then builds and publishes client-side application to S3 and Cloud Front
awsmobile run (p. 390)	Executes `awsmobile push`, then executes the project's start command to test run the client-side application
awsmobile console (p. 390)	Open the web console of the awsmobile Mobile Hub project in the default browser
awsmobile features (p. 390)	Shows available and enabled features. Toggle to select or de-select features.
awsmobile <feature-name> enable [--prompt] (p. 391)	Enables the feature with the defaults (and prompt for changes)
awsmobile <feature-name> disable (p. 392)	Disables the feature
awsmobile <feature-name> configure (p. 393)	Contains feature-specific sub commands like add-table, add-api, etc.
awsmobile cloud-api invoke <apiname> <method> <path> [init] (p. 394)	Invokes the API for testing locally. This helps quickly test unsigned APIs in your local environment.
awsmobile delete (p. 394)	Deletes the Mobile hub project.
awsmobile help [cmd] (p. 395)	Displays help for [cmd].

init

The `awsmobile init` command initializes a new Mobile Hub project, checks for IAM keys, and pulls the aws-exports.js file.

There are two usages of the `awsmobile init` command

1. Initialize the current project with awsmobilejs features

```
awsmobile init
```

When prompted, set these project configs:

```
Please tell us about your project:
? Where is your project's source directory:  src
```

```
? Where is your project's distribution directory that stores build artifacts:  build
? What is your project's build command:  npm run-script build
? What is your project's start command for local test run:  npm run-script start

? What awsmobile project name would you like to use:  my-mobile-project
```

The source directory is where the the AWS Mobile CLI copies the latest `aws-exports.js` to be easily available for your front-end code. This file is automatically updated everytime features are added or removed. Specifying a wrong / unavailable folder will not copy the file over.

The Distribution directly is essentially the build directory for your project. This is used during the `awsmobile publish` process.

The project's build and start values are used during the `awsmobile publish` and `awsmobile run` commands respectively.

The awsmobile project name is the name of the backend project created in the Mobile hub.

You can alter the settings about your project by using the awsmobile configure project (p. 388) command.

2. Initialize and link to an existing awsmobile project as backend

```
awsmobile init <awsmobile-project-id>
```

The awsmobile-project-id is the id of the existing backend project in the Mobile Hub. This command helps attach an existing backend project to your app.

3. Remove the attached awsmobile project from the backend.

```
awsmobile init --remove
```

This command removes the attached backend project associated with your app and cleans the associated files. This will not alter your app in any way, other than removing the backend project itself.

configure

The `awsmobile configure` shows existing keys and allows them to be changed if already set. If keys aren't set, deep links the user to the IAM console to create keys and then prompts for the access key and secret key. There are two possible usages of this command. Based on the argument selected, this command can be used to set or change the aws account settings OR the project settings.

```
awsmobile configure [aws|project]
```

1. Configuring the aws account settings using the `aws` argument. This is the default argument for this command

```
awsmobile configure
or
awsmobile configure aws
```

You will be prompted with questions to set the aws account credentials as below

```
configure aws
? accessKeyId:  <ACCESS-KEY-ID>
? secretAccessKey:  <SECRET-ACCESS-KEY>
```

```
? region:  <SELECT-REGION-FROM-THE-LIST>
```

2. Configuring the project settings using the `project` argument

```
awsmobile configure project
```

You will be prompted with questions to configure project as detailed below

```
? Where is your project's source directory:  src
? Where is your project's distribution directory to store build artifacts:  dist
? What is your project's build command:  npm run-script build
? What is your project's start command for local test run:  npm run-script start
```

3. Retrieve and display the aws credentials using the `--list` option

```
awsmobile configure --list
```

pull

The `awsmobile pull` command downloads the latest aws-exports.js, YAML and any relevant cloud / backend artifacts from the Mobile Hub project to the local dev environment. Use this command if you modified the project on the Mobile Hub and want to get the latest on your local environment.

```
awsmobile pull
```

push

The `awsmobile push` uploads local metadata, Lambda code, Dynamo definitions and any relevant artifacts to Mobile Hub. Use this command when you enable, disable or configure features on your local evironment and want to update the backend project on the Mobile Hub with the relevant updates.

```
awsmobile push
```

Use `awsmobile push` after using `awsmobile features`, `awsmobile <feature> enable`, `awsmobile <feature> disable` or `awsmobile <feature> configure` to update the backend project appropriately. This can be used either after each of these or once after all of the changes are made locally.

publish

The `awsmobile publish` command first executes the awsmobile `push` command, then builds and publishes client-side code to Amazon S3 hosting bucket. This command publishes the client application to s3 bucket for hosting and then opens the browser to show the index page. It checks the timestamps to automatically build the app if necessary before deployment. It checks if the client has selected hosting in their backend project features, and if not, it'll prompt the client to update the backend with hosting feature.

```
awsmobile publish
```

The publish command has a number of options to be used.

1. Refresh the Cloud Front distributions

```
awsmobile publish -c
 or
awsmobile publish --cloud-front
```

2. Test the application on AWS Device Farm

```
awsmobile publish -t
or
awsmobile publish --test
```

3. Suppress the tests on AWS Device Farm

```
awsmobile publish -n
```

4. Publish the front end only without updating the backend

```
awsmobile publish -f
or
awsmobile publish --frontend-only
```

run

The `awsmobile run` command first executes the `awsmobile push` command, then executes the start command you set in the project configuration, such as `npm run start` or `npm run ios`. This can be used to conveniently test run your application locally with the latest backend development pushed to the cloud.

```
awsmobile run
```

console

The `awsmobile console` command opens the web console of the awsmobile Mobile Hub project in the default browser

```
awsmobile console
```

features

The `awsmobile features` command displays all the available awsmobile features, and allows you to individually enable/disable them locally. Use the arrow key to scroll up and down, and use the space key to enable/disable each feature. Please note that the changes are only made locally, execute awsmobile push to update the awsmobile project in the cloud.

```
awsmobile features
```

The features supported by the AWS Mobile CLI are:

- user-signin (Amazon Cognito)
- user-files (Amazon S3)
- cloud-api (Lambda / API Gateway)
- database (DynamoDB)
- analytics (Amazon Pinpoint)

- hosting (Amazon S3 and CloudFront)

```
? select features:  (Press <space> to select, <a> to toggle all, <i> to inverse selection)
## user-signin
 # user-files
 # cloud-api
 # database
 # analytics
 # hosting
```

Use caution when disabling a feature. Disabling the feature will delete all the related objects (APIs, Lambda functions, tables etc). These artifacts can not be recovered locally, even if you re-enable the feature.

Use `awsmobile push` after using `awsmobile <feature> disable` to update the backend project on the AWS Mobile Hub project with the selected features.

enable

The `awsmobile <feature> enable` enables the specified feature with the default settings. Please note that the changes are only made locally, execute `awsmobile` push to update the AWS Mobile project in the cloud.

```
awsmobile <feature> enable
```

The features supported by the AWS Mobile CLI are:

- user-signin (Amazon Cognito)
- user-files (Amazon S3)
- cloud-api (Lambda / API Gateway)
- database (DynamoDB)
- analytics (Amazon Pinpoint)
- hosting (Amazon S3 and CloudFront)

The `awsmobile <feature> enable --prompt` subcommand allows user to specify the details of the mobile hub feature to be enabled, instead of using the default settings. It prompts the user to answer a list of questions to specify the feature in detail.

```
awsmobile <feature> enable -- prompt
```

Enabling the `user-signin` feature will prompt you to change the way it is enabled, configure advanced settings or disable sign-in feature to the project. Selecting the desired option may prompt you with further questions.

```
awsmobile user-signin enable --prompt

? Sign-in is currently disabled, what do you want to do next (Use arrow keys)
# Enable sign-in with default settings
  Go to advance settings
```

Enabling the `user-files` feature with the `--prompt` option will prompt you to confirm usage of S3 for user files.

```
awsmobile user-files enable --prompt
```

```
? This feature is for storing user files in the cloud, would you like to enable it? Yes
```

Enabling the `cloud-api` feature with the `--prompt` will prompt you to create, remove or edit an API related to the project. Selecting the desired option may prompt you with further questions.

```
awsmobile cloud-api enable --prompt

 ? Select from one of the choices below. (Use arrow keys)
 # Create a new API
```

Enabling the `database` feature with the `--prompt` will prompt you to with initial questions to specify your database table details related to the project. Selecting the desired option may prompt you with further questions.

```
awsmobile database enable --prompt

? Should the data of this table be open or restricted by user? (Use arrow keys)
# Open
  Restricted
```

Enabling the `analytics` feature with the `--prompt` will prompt you to confirm usage of Pinpoint Analytics.

```
 awsmobile analytics enable --prompt

? Do you want to enable Amazon Pinpoint analytics? (y/N)
```

Enabling the `hosting` feature with the `--prompt` will prompt you to confirm hosting and streaming on CloudFront distribution.

```
awsmobile hosting enable --prompt

? Do you want to host your web app including a global CDN? (y/N)
```

Execute `awsmobile push` after using `awsmobile <feature> enable` to to update the awsmobile project in the cloud.

disable

The `awsmobile <feature> disable` disables the feature in their backend project. Use caution when disabling a feature. Disabling the feature will delete all the related objects (APIs, Lambda functions, tables etc). These artifacts can not be recovered locally, even if you re-enable the feature.

```
awsmobile <feature> disable
```

The features supported by the AWS Mobile CLI are:

- user-signin (Amazon Cognito)
- user-files (Amazon S3)
- cloud-api (Lambda / API Gateway)
- database (DynamoDB)
- analytics (Amazon Pinpoint)
- hosting `

Use `awsmobile push` after using `awsmobile <feature> disable` to update the backend project on the AWS Mobile Hub project with the disabled features.

configure

The `awsmobile <feature> configure` configures the objects in the selected feature. The configuration could mean adding, deleting or updating a particular artifact. This command can be used only if the specfic feature is already enabled.

```
awsmobile <feature> configure
```

The features supported by the AWS Mobile CLI are:

- user-signin (Amazon Cognito)
- user-files (Amazon S3)
- cloud-api (Lambda / API Gateway)
- database (DynamoDB)
- analytics (Amazon Pinpoint)
- hosting (Amazon S3 and CloudFront)

Configuring the `user-signin` feature will prompt you to change the way it is enabled, configure advanced settings or disable sign-in feature to the project. Selecting the desired option may prompt you with further questions.

```
awsmobile user-signin configure

? Sign-in is currently enabled, what do you want to do next (Use arrow keys)
# Configure Sign-in to be required (Currently set to optional)
  Go to advance settings
  Disable sign-in
```

Configuring the `user-files` feature will prompt you to confirm usage of S3 for user files.

```
 awsmobile user-files configure

? This feature is for storing user files in the cloud, would you like to enable it? (Y/n)
```

Configuring the `cloud-api` feature will prompt you to create, remove or edit an API related to the project. Selecting the desired option may prompt you with further questions.

```
awsmobile cloud-api configure

? Select from one of the choices below. (Use arrow keys)
# Create a new API
  Remove an API from the project
  Edit an API from the project
```

Configuring the `database` feature will prompt you to create, remove or edit a table related to the project. Selecting the desired option may prompt you with further questions.

```
awsmobile database configure

  ? Select from one of the choices below. (Use arrow keys)
  # Create a new table
```

```
    Remove table from the project
    Edit table from the project
```

Configuring the `analytics` feature will prompt you to confirm usage of Pinpoint Analytics.

```
awsmobile analytics configure

? Do you want to enable Amazon Pinpoint analytics? Yes
```

Configuring the `hosting` feature will prompt you to confirm hosting and streaming on CloudFront distribution.

```
awsmobile hosting configure

? Do you want to host your web app including a global CDN? Yes
```

Use `awsmobile push` after using `awsmobile <feature> configure` to update the backend project on the AWS Mobile Hub project with the configured features.

invoke

The `awsmobile cloud-api invoke` invokes the API for testing locally. This helps quickly test the unsigned API locally by passing the appropritate arguments. This is intended to be used for the development environment or debugging of your API / Lambda function.

```
awsmobile cloud-api invoke <apiname> <method> <path> [init]
```

For example you could invoke the sampleCloudApi post method as shown below

```
awsmobile cloud-api invoke sampleCloudApi post /items '{"body":{"test-key":"test-value"}}'
```

The above test will return a value that looks like

```
{ success: 'post call succeed!',
  url: '/items',
  body: { 'test-key': 'test-value' } }
```

Similarly, you could invoke the sampleCloudApi get method as shown below

```
awsmobile cloud-api invoke sampleCloudApi get /items
```

The above test will return a value that looks like

```
{ success: 'get call succeed!', url: '/items' }
```

delete

The `awsmobile delete` command deletes the Mobile hub project in the cloud. Use extra caution when you decide to execute this command, as it can irrevocably affect your team's work, the mobile hub project will be delete and cannot be recovered once this command is executed.

```
awsmobile delete
```

help

The `awsmobile help` command can be used as a standalone command or the command name that you need help in can be passed as an argument. This gives the usage information for that command including any options that can be used with it.

For Example:

```
awsmobile help
or
awsmobile help init
```

The `--help` option detailing at the beginning of this page and the `awsmobile help` command provide the same level of detail. The difference is in the usage.

AWS Mobile CLI User Credentials

Overview

As described on the AWS Mobile CLI Get Started (p. 366) page, the first time you set up the CLI you will be prompted to provide AWS user credentials. The credentials establish permissions for the CLI to manage AWS services on your behalf. They must belong to an AWS IAM user with administrator permissions in the account where the CLI is being used.

Permissions

Administrator permissions are granted by an AWS account administrator. If don't have administrator permissions you will need to ask an administrator for the AWS account to grant them.

If you are the account owner and signed in under the root credentials for the account, then you have, or can grant yourself, administrator permissions using the `AdministratorAccess` managed policy. Best practice is to create a new IAM user under your account to access AWS services instead of using root credentials.

For more information, see Control Access to Mobile Hub Projects (p. 314).

Get Account User Credentials

If you have administrator permissions, the values you need to provide the CLI are your IAM user's Access Key ID and a Secret Access Key. If not, you will need to get these from an administrator.

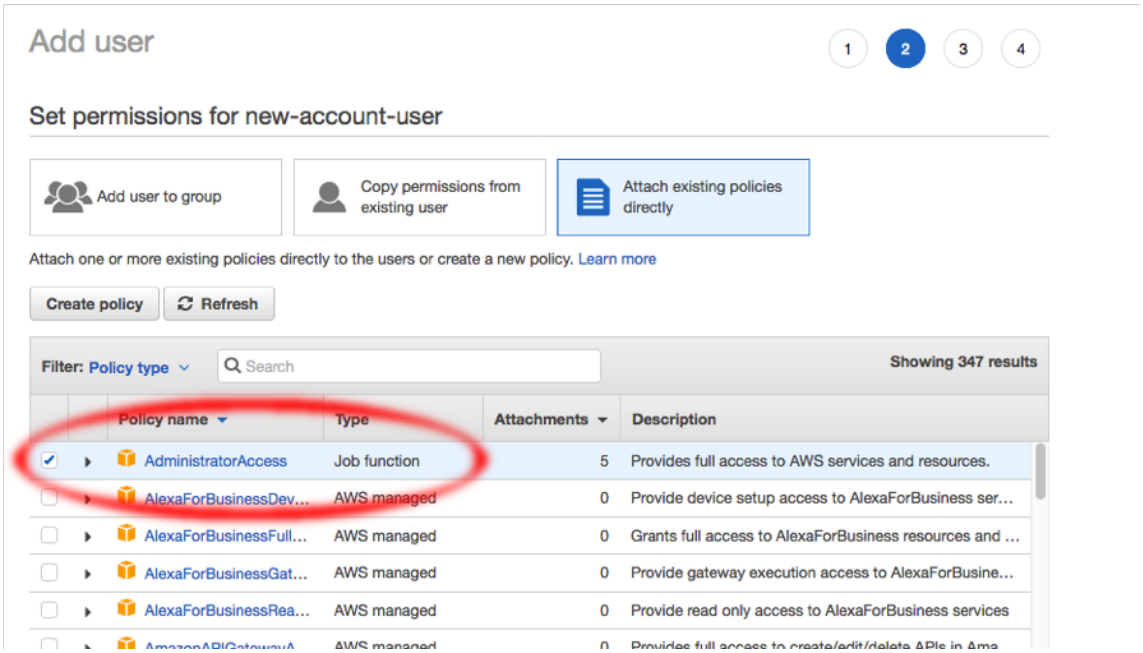
To provide the ID and the Key to AWS CLI, follow the CLI prompts to sign-in to AWS, and provide a user name and AWS region. The CLI will open the AWS IAM console **Add user** dialog, with the `AdministratorAccess` policy attached, and the **Programmatic access** option selected by default.

Topics

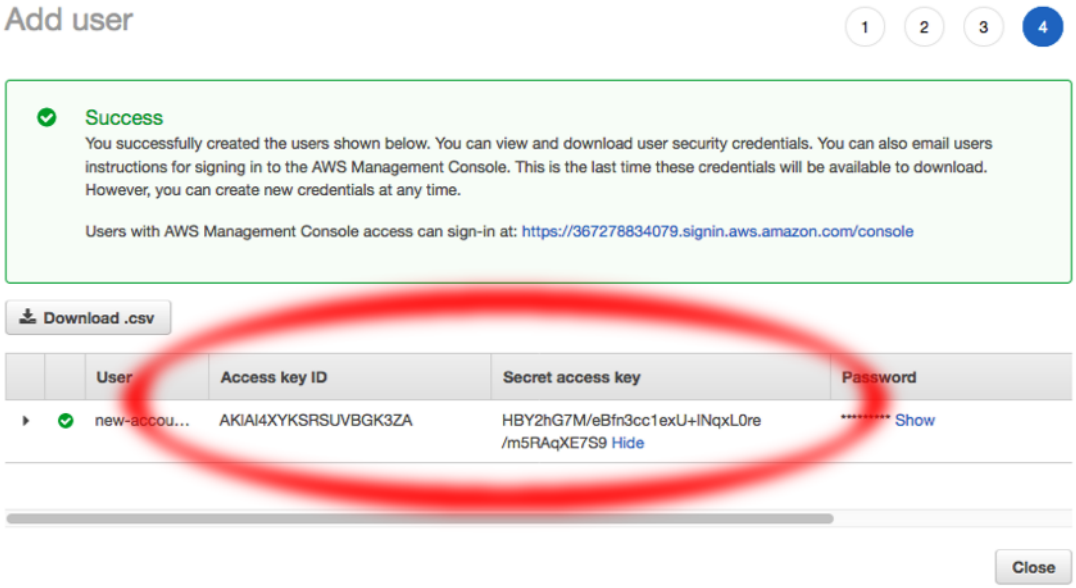
Get credentials for a new user

1. Choose **Next: Permissions** and then choose **Create user**.

   Alternatively, you could add the user to a group with `AdministratorAccess` attached.



2. Choose **Create user**.
3. Copy the values from the table displayed, or choose **Download .csv** to save the values locally, and then type them into the prompts.



For more detailed steps, see add a new account user with administrator permissions (p. 315).

Get credentials for an existing user

1. Choose **cancel**.
2. On the left, choose **Users**, then select the user from the list. Choose **Security credentials**, then choose **Create access key**.

AWS Amplify Library for React Native

AWS Amplify is an open source JavaScript library for frontend and mobile developers building cloud-enabled applications. The library is a declarative interface across different categories of operations in order to make common tasks easier to add into your application. The default implementation works with Amazon Web Services (AWS) resources but is designed to be open and pluggable for usage with other cloud services that wish to provide an implementation or custom backends.

The AWS Mobile CLI, built on AWS Mobile Hub, provides a command line interface for frontend JavaScript developers to seamlessly enable and configure AWS services into their apps. With minimal configuration, you can start using all of the functionality provided by the AWS Mobile Hub from your favorite terminal application.

Topics

Get Started

Overview

The AWS Mobile CLI provides a command line experience that allows frontend JavaScript developers to quickly create and integrate AWS backend resources into their mobile apps.

Prerequisites

1. Sign up for the AWS Free Tier to learn and prototype at little or no cost.
2. Install Node.js with NPM.
3. Install the AWS Mobile CLI

```
npm install --global awsmobile-cli
```

4. Configure the CLI with your AWS credentials

   To setup permissions for the toolchain used by the CLI, run:

   ```
   awsmobile configure
   ```

   If prompted for credentials, follow the steps provided by the CLI. For more information, see Provide IAM credentials to AWS Mobile CLI (p. 395).

Set Up Your Backend

Need to create a quick sample React Native app? See Create a React Native App.

To configure backend features for your app

1. In the root folder of your app, run:

```
awsmobile init
```

   The `init` command creates a backend project for your app. By default, analytics and web hosting are enabled in your backend and this configuration is automatically pulled into your app when you initialize.
2. When prompted, provide the source directory for your project. The CLI will generate `aws-exports.js` in this location. This file contains the configuration and endpoint metadata used to link your frontend to your backend services.

```
? Where is your projects's source directory:  /
```

   Then respond to further prompts with the following values.

```
Please tell us about your project:
? Where is your project's source directory:  /
? Where is your project's distribution directory that stores build artifacts:  build
? What is your project's build command:  npm run-script build
? What is your project's start command for local test run:  npm run-script start
```

Connect to Your Backend

AWS Mobile uses the open source AWS Amplify library to link your code to the AWS features configured for your app.

To connect the app to your configured AWS services

1. Install AWS Amplify for React Native library.

```
npm install --save aws-amplify
```

2. In `App.js` (or in other code that runs at launch-time), add the following imports.

```
import Amplify from 'aws-amplify';

import aws_exports from './YOUR-PATH-TO/aws-exports';
```

3. Then add the following code.

```
Amplify.configure(aws_exports);
```

Run Your App Locally

Your app is now ready to launch and use the default services configured by AWS Mobile.

To launch your app locally

Use the command native to the React Native tooling you are using. For example, if you made your app using `create-react-native-app` then run:

```
npm run android
```

```
# OR

npm run ios
```

Anytime you launch your app, app usage analytics are gathered and can be visualized (p. 399) in an AWS console.

AWS Free Tier	Initializing your app or adding features through the CLI will cause AWS services to be configured on your behalf. The pricing for AWS Mobile services enables you to learn and prototype at little or no cost using the AWS Free Tier.

Next Steps

Add Features

Add the following AWS Mobile features to your mobile app using the CLI.

- Analytics (p. 399)
- User Sign-in (p. 401)
- NoSQL Database (p. 402)
- User File Storage (p. 406)
- Cloud Logic (p. 408)

Learn more

To learn more about the commands and usage of the AWS Mobile CLI, see the AWS Mobile CLI reference (p. 386).

Learn about AWS Mobile Amplify.

Add Analytics

BEFORE YOU BEGIN	The steps on this page assume you have already completed the steps on Get Started (p. 397).

Basic Analytics Backend is Enabled for Your App

When you complete the AWS Mobile CLI setup and launch your app, anonymized session and device demographics data flows to the AWS analytics backend.

To send basic app usage analytics to AWS

Launch your app locally, for instance, if you created your app using `create-react-native-app`, by running:

```
npm run android
```

```
# Or

npm run ios
```

When you use your app the Amazon Pinpoint service gathers and visualizes analytics data.

To view the analytics using the Amazon Pinpoint console

1. Launch your app at least once.
2. Open your project in the AWS Mobile Hub console.

```
awsmobile console
```

3. Choose the **Analytics** icon on the left, to navigate to your project in the Amazon Pinpoint console.
4. Choose **Analytics** on the left.

You should see an up-tick in several graphs.

Add Custom Analytics to Your App

You can configure your app so that Amazon Pinpoint gathers data for custom events that you register within the flow of your code.

To instrument custom analytics in your app

In the file containing the event you want to track, add the following import:

```
import { Analytics } from 'aws-amplify';
```

Add the a call like the following to the spot in your JavaScript where the tracked event should be fired:

```
componentDidMount() {
   Analytics.record('FIRST-EVENT-NAME');
}
```

Or to relevant page elements:

```
handleClick = () => {
     Analytics.record('SECOND-EVENT-NAME');
}

<Button title="Record event" onPress={this.handleClick}/>
```

To test:

1. Save the changes and launch your app. Use your app so that tracked events are triggered.
2. In the Amazon Pinpoint console, choose **Events** near the top.
3. Select an event in the **Event** dropdown menu on the left.

Custom event data may take a few minutes to become visible in the console.

Next Steps

Learn more about the analytics in AWS Mobile which are part of the Messaging and Analytics (p. 340) feature. This feature uses Amazon Pinpoint.

Learn about AWS Mobile CLI (p. 386).

Learn about the AWS Amplify for React Native library.

Add Auth / User Sign-in

BEFORE YOU BEGIN	The steps on this page assume you have already completed the steps on Get Started (p. 397).

Set Up Your Backend

The AWS Mobile CLI components for user authentication include a rich, configurable UI for sign-up and sign-in.

To enable the Auth features

In the root folder of your app, run:

```
awsmobile user-signin enable

awsmobile push
```

Connect to Your Backend

The AWS Mobile CLI enables you to integrate ready-made sign-up/sign-in/sign-out UI from the command line.

To add user auth UI to your app

1. Install AWS Amplify for React Nativelibrary.

```
npm install --save aws-amplify
npm install --save aws-amplify-react-native
```

Note	If your react-native app was not created using `create-react-native-app` or using a version of Expo lower than v25.0.0 (the engine behind `create-react-native-app`), you will need to link libraries in your project for the Auth module on React Native, `amazon-cognito-identity-js`.
	
	To link to the module, you must first eject the project:
	
	`npm run eject`
	`react-native link amazon-cognito-identity-js`

1. Add the following import in `App.js` (or other file that runs upon app startup):

```
import { withAuthenticator } from 'aws-amplify-react-native';
```

2. Then change `export default App;` to the following.

```
export default withAuthenticator(App);
```

To test, run `npm start` or `awsmobile run`.

Next Steps

Learn more about the AWS Mobile User Sign-in (p. 348) feature, which uses Amazon Cognito.

Learn about AWS Mobile CLI (p. 386).

Learn about AWS Mobile Amplify.

Access Your Database

BEFORE YOU BEGIN	The steps on this page assume you have already completed the steps on Get Started (p. 397).

Set Up Your Backend

AWS Mobile `database` feature enables you to create tables customized to your needs. The CLI then guides you to create a custom API to access your database.

Create a table

To specify and create a table

1. In your app root folder, run:

```
awsmobile database enable --prompt
```

2. Design your table when prompted by the CLI.

   The CLI will prompt you for the table and other table configurations such as columns.

```
Welcome to NoSQL database wizard
You will be asked a series of questions to help determine how to best construct your
 NoSQL database table.

? Should the data of this table be open or restricted by user? Open
? Table name Notes


You can now add columns to the table.

? What would you like to name this column NoteId
? Choose the data type string
? Would you like to add another column Yes
? What would you like to name this column NoteTitle
? Choose the data type string
```

```
? Would you like to add another column Yes
? What would you like to name this column NoteContent
? Choose the data type string
? Would you like to add another column No
```

Choose a Primary Key that will uniquely identify each item. Optionally, choose a column to be a Sort Key when you will commonly use those values in combination with the Primary Key for sorting or searching your data. You can additional sort keys by adding a Secondary Index for each column you will want to sort by.

```
Before you create the database, you must specify how items in your table are uniquely
 organized. This is done by specifying a Primary key. The primary key uniquely identifies
 each item in the table, so that no two items can have the same key.
This could be and individual column or a combination that has "primary key" and a "sort
 key".
To learn more about primary key:
http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
HowItWorks.CoreComponents.html#HowItWorks.CoreComponents.PrimaryKey


? Select primary key NoteId
? Select sort key (No Sort Key)

You can optionally add global secondary indexes for this table. These are useful when
 running queries defined by a different column than the primary key.

To learn more about indexes:

http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
HowItWorks.CoreComponents.html#HowItWorks.CoreComponents.SecondaryIndexes

? Add index No
Table Notes added
```

Create a CRUD API

AWS Mobile will create a custom API for your app to perform create, read, update, and delete (CRUD) actions on your database.

To create a CRUD API for your table

1. In the root folder of your app, run:

```
awsmobile cloud-api enable --prompt
```

2. When prompted, choose `Create CRUD API for existing Dynamo table`, select the table name from the previous steps, choose the access permissions for the table. Using the example table from the previous section:

```
? Select from one of the choices below.
  Create a new API
# Create CRUD API for an existing Amazon DynamoDB table
```

The prompt response will be:

```
Path to be used on API for get and remove an object should be like:
/Notes/object/:NoteId

Path to be used on API for list objects on get method should be like:
```

```
/Notes/:NoteId

JSON to be used as data on put request should be like:
{
  "NoteTitle": "INSERT VALUE HERE",
  "NoteContent": "INSERT VALUE HERE",
  "NoteId": "INSERT VALUE HERE"
}
To test the api from the command line (after awsmobile push) use this commands
awsmobile cloud-api invoke NotesCRUD <method> <path> [init]
Api NotesCRUD saved
```

Copy and keep the path of your API and the JSON for use in your app code.

This feature will create an API using Amazon API Gateway and AWS Lambda. You can optionally have the lambda function perform CRUD operations against your Amazon DynamoDB table.

3. Update your backend.

   To create the API you have configured, run:

```
awsmobile push
```

   Until deployment of API to the cloud the has completed, the CLI displays the message: `cloud-api update status: CREATE_IN_PROGRESS`. Once deployed a sucessful creation message `cloud-api update status: CREATE_COMPLETE` is displayed.

   You can view the API that the CLI created by running `awmobile console` and then choosing **Cloud Logic** in the Mobile Hub console.

Connect to Your Backend

Topics

To access to database tables from your app

1. In `App.js` import the following.

```
import Amplify, { API } from 'aws-amplify';
import aws_exports from 'path_to_your_aws-exports';
Amplify.configure(aws_exports);
```

2. Add the following `state` to your component.

```
state = {
  apiResponse: null,
  noteId: ''
     };

  handleChangeNoteId = (event) => {
        this.setState({noteId: event});
}
```

Save an item (create or update)

To save an item

In the part of your app where you access the database, such as an event handler in your React component, call the `put` method. Use the JSON and the root path (`/Notes`) of your API that you copied from the CLI prompt response earlier.

```
// Create a new Note according to the columns we defined earlier
  async saveNote() {
    let newNote = {
      body: {
        "NoteTitle": "My first note!",
        "NoteContent": "This is so cool!",
        "NoteId": this.state.noteId
      }
    }
    const path = "/Notes";

    // Use the API module to save the note to the database
    try {
      const apiResponse = await API.put("NotesCRUD", path, newNote)
      console.log("response from saving note: " + apiResponse);
      this.setState({apiResponse});
    } catch (e) {
      console.log(e);
    }
  }
```

To use the command line to see your saved items in the database run:

```
awsmobile cloud-api invoke NotesCRUD GET /Notes/object/${noteId}
```

Get a specific item

To query for a specific item

Call the `get` method using the API path (copied earlier) to the item you are querying for.

```
// noteId is the primary key of the particular record you want to fetch
    async getNote() {
      const path = "/Notes/object/" + this.state.noteId;
      try {
        const apiResponse = await API.get("NotesCRUD", path);
        console.log("response from getting note: " + apiResponse);
        this.setState({apiResponse});
      } catch (e) {
        console.log(e);
      }
    }
```

Delete an item

To delete an item

Add this method to your component. Use your API path (copied earlier).

```
// noteId is the NoteId of the particular record you want to delete
```

```
    async deleteNote() {
      const path = "/Notes/object/" + this.state.noteId;
      try {
        const apiResponse = await API.del("NotesCRUD", path);
        console.log("response from deleteing note: " + apiResponse);
        this.setState({apiResponse});
      } catch (e) {
        console.log(e);
      }
    }
```

UI to exercise CRUD calls

The following is and example of how you might construct UI to excercise these operations.

```
<View style={styles.container}>
        <Text>Response: {this.state.apiResponse &&
 JSON.stringify(this.state.apiResponse)}</Text>
        <Button title="Save Note" onPress={this.saveNote.bind(this)} />
        <Button title="Get Note" onPress={this.getNote.bind(this)} />
        <Button title="Delete Note" onPress={this.deleteNote.bind(this)} />
        <TextInput style={styles.textInput} autoCapitalize='none'
 onChangeText={this.handleChangeNoteId}/>
</View>

const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: '#fff',
    alignItems: 'center',
    justifyContent: 'center',
  },
  textInput: {
      margin: 15,
      height: 30,
      width: 200,
      borderWidth: 1,
      color: 'green',
      fontSize: 20,
      backgroundColor: 'black'
   }
});
```

Next Steps

Learn more about the AWS Mobile NoSQL Database (p. 335) feature, which uses Amazon DynamoDB.

Learn about AWS Mobile CLI (p. 386).

Learn about AWS Mobile Amplify.

Add Storage

BEFORE YOU BEGIN	The steps on this page assume you have already completed the steps on Get Started (p. 397).

The AWS Mobile CLI User File Storage (p. 353) feature enables apps to store user files in the cloud.

Set Up Your Backend

To configure your app's cloud storage location

In your app root folder, run:

```
awsmobile user-files enable

awsmobile push
```

Connect to Your Backend

To add User File Storage to your app

In your component where you want to transfer files:

Import the `Storage` module from aws-amplify and configure it to communicate with your backend.

```
import { Storage } from 'aws-amplify';
```

Now that the Storage module is imported and ready to communicate with your backend, implement common file transfer actions using the code below.

Upload a file

To upload a file to storage

Add the following methods to the component where you handle file uploads.

```
async uploadFile() {
  let file = 'My upload text';
  let name = 'myFile.txt';
  const access = { level: "public" }; // note the access path
  Storage.put(name, file, access);
}
```

Get a specific file

To download a file from cloud storage

Add the following code to a component where you display files.

```
async getFile() {
  let name = 'myFile.txt';
  const access = { level: "public" };
  let fileUrl = await Storage.get(name, access);
  // use fileUrl to get the file
}
```

List all files

To list the files stored in the cloud for your app

Add the following code to a component where you list a collection of files.

```
async componentDidMount() {
  const path = this.props.path;
  const access = { level: "public" };
  let files = await Storage.list(path, access);
   // use file list to get single files
}
```

Use the following code to fetch file attributes such as the size or time of last file change.

```
file.Size; // file size
file.LastModified.toLocaleDateString(); // last modified date
file.LastModified.toLocaleTimeString(); // last modified time
```

Delete a file

Add the following state to the element where you handle file transfers.

```
async deleteFile(key) {
  const access = { level: "public" };
  Storage.remove(key, access);
}
```

Next Steps

Learn more about the analytics in AWS Mobile which are part of the User File Storage (p. 353) feature. This feature uses Amazon Simple Storage Service (S3).

Learn about AWS Mobile CLI (p. 386).

Learn about AWS Mobile Amplify.

Access Your APIs

BEFORE YOU BEGIN	The steps on this page assume you have already completed the steps on Get Started (p. 397).

Set Up Your Backend

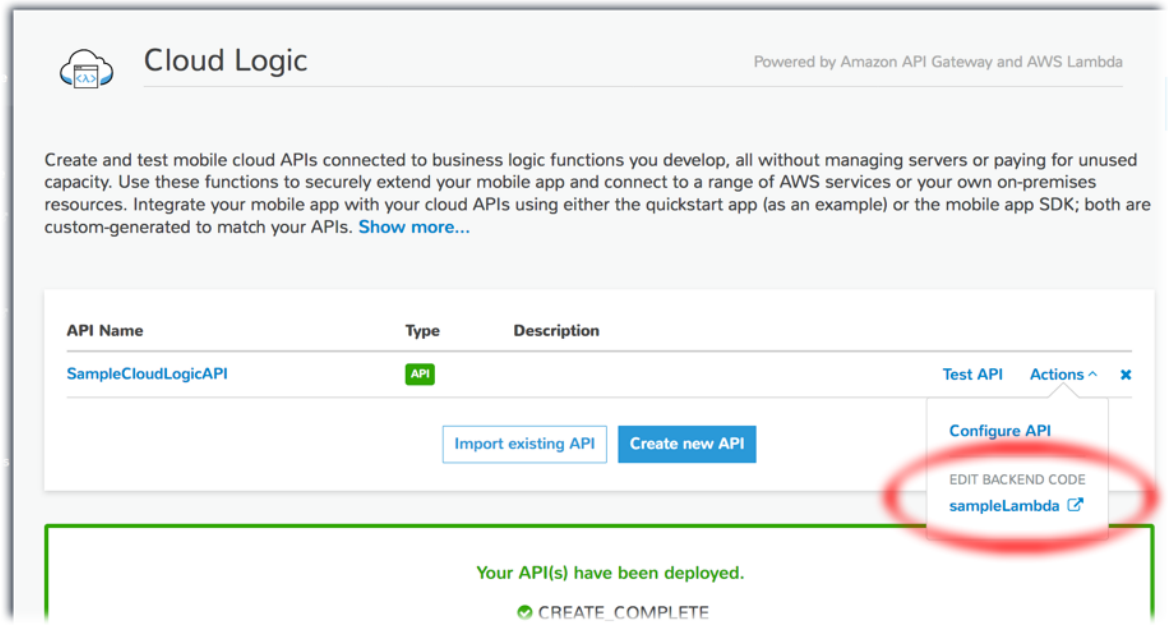
The AWS Mobile Cloud Logic (p. 332) feature lets you call APIs in the cloud. API calls are handled by your serverless Lambda functions.

To enable cloud APIs in your app

```
awsmobile cloud-api enable

awsmobile push
```

Enabling Cloud Logic in your app adds a sample API, `sampleCloudApi` to your project that can be used for testing.

You can find the sample handler function for the API by running `awsmobile console` in your app root folder, and then choosing the **Cloud Logic** feature in your Mobile Hub project.



Quickly Test Your API From the CLI

The `sampleCloudApi` and its handler function allow you to make end to end API calls.

To test invocation of your unsigned APIs in the development environment

```
awsmobile cloud-api invoke <apiname> <method> <path> [init]
```

For the `sampleCloudApi` you may use the following examples to test the `post` method

```
awsmobile cloud-api invoke sampleCloudApi post /items '{"body": {"testKey":"testValue"}}'
```

This call will return a response similar to the following.

```
{ success: 'post call succeed!',
url: '/items',
body: { testKey: 'testValue' } }
```

To test the :get method

```
awsmobile cloud-api invoke sampleCloudApi get /items
```

This will return a response as follows.

```
{ success: 'get call succeed!', url: '/items' }
```

Connect to Your Backend

Once you have created your own Cloud Logic (p. 332) APIs and Lambda functions, you can call them from your app.

To call APIs from your app

In `App.js` (or other code that runs at launch-time), add the following import.

```
import Amplify, { API } from 'aws-amplify';
import aws_exports from './aws-exports';
Amplify.configure(aws_exports);
```

Then add this to the component that calls your API.

```
state = { apiResponse: null };

async getSample() {
 const path = "/items"; // you can specify the path
  const apiResponse = await API.get("sampleCloudApi" , path); //replace the API name
  console.log('response:' + apiResponse);
  this.setState({ apiResponse });
}
```

To invoke your API from a UI element, add an API call from within your component's `render()` method.

```
<View>
   <Button title="Send Request" onPress={this.getSample.bind(this)} />
   <Text>Response: {this.state.apiResponse && JSON.stringify(this.state.apiResponse)}</
Text>
</View>
```

To test, save the changes, run `npm run android` or `npm run ios` to launch your app. Then try the UI element that calls your API.

Next Steps

Learn more about the AWS Mobile Cloud Logic (p. 332) feature which uses Amazon API Gateway and AWS Lambda.

To be guided through creation of an API and it's handler, run `awsmobile console` to open your app in the Mobile Hub console, and choose **Cloud Logic**.

Learn about AWS Mobile CLI (p. 386).

Learn about AWS Mobile Amplify.

AWS Mobile Hub Features

The following pages contain reference material for the AWS Mobile CLI for Web (JavaScript).

- AWS Mobile CLI Reference (p. 386)
- AWS Mobile CLI Credentials (p. 395)