AWS Mobile SDK Android Developer Guide

Release 0.0.3

Amazon Web Services

Jan 18, 2017

Contents

Welcome to the *AWS Mobile SDK for Android Developer Guide*. This guide will help you start developing Android applications using Amazon Web Services.

If you're new to the AWS Mobile SDK, you'll probably want to look first at *What is the AWS Mobile SDK for Android?* and *Getting Started with the AWS Mobile SDK for Android*. These topics explain what the AWS Mobile SDK includes, how to set up the SDK, and how to get started using AWS services from an Android application.

CHAPTER 1

What is the AWS Mobile SDK for Android?

The AWS Mobile SDK for Android is an open-source software development kit distributed under the Apache Open Source license. The SDK provides libraries, code samples, and documentation to help developers build connected mobile applications using Amazon Web Services (AWS). Supported AWS services currently include:

- Amazon Cognito Identity
- Amazon Cognito Sync
- Mobile Analytics
- Amazon S3
- DynamoDB
- Amazon Kinesis
- Lambda
- *Amazon Lex*
- *Amazon Polly*
- *Amazon Pinpoint*

1.1 Compatibility

The AWS Mobile SDK for Android is compatible with Android 2.3.3 (API Level 10) or higher. For more information about the Android platform, see Android Developers.

1.2 Download the AWS Mobile SDK for Android

- Download AWS Mobile SDK for Android (zip file)
- Source on Github

1.3 About the AWS Mobile Services

The AWS Mobile SDKs include client-side libraries for working with Amazon Web Services. These client libraries provide high-level, mobile-optimized interfaces to services such as DynamoDB, Amazon S3, and Amazon Kinesis.

The Mobile SDKs also include clients for Amazon Cognito and Amazon Mobile Analytics—web services designed specifically for use by mobile developers.

1.3.1 Amazon Cognito

Amazon Cognito facilitates the delivery of scoped, temporary credentials to mobile devices or other untrusted environments, and it uniquely identifies a device or user and supplies the user with a consistent identity throughout the lifetime of an application. The Amazon Cognito Sync service enables cross-device syncing of application-related user data. Amazon Cognito also persists data locally, so that it's available even if the device is offline.

After you set up the SDK, you can start using Amazon Cognito by following the instructions at *Authenticate Users with Amazon Cognito Identity* and *Sync User Data with Amazon Cognito Sync*.

1.3.2 Amazon Mobile Analytics

Amazon Mobile Analytics lets you collect, visualize, and understand app usage for your mobile apps. Reports are available for metrics on active users, sessions, retention, in-app revenue, and custom events, and can be filtered by platform and date range.

After you set up the SDK, you can start using Amazon Mobile Analytics by following the instructions at *Track App Usage Data with Amazon Mobile Analytics*.

1.3.3 Amazon Simple Storage Service (S3)

Amazon Simple Storage Service (Amazon S3) provides secure, durable, highly-scalable object storage in the cloud. Using the AWS Mobile SDK, you can directly access Amazon S3 from your mobile app.

After you set up the SDK, you can start using Amazon S3 by following the instructions at *Store and Retrieve Files with Amazon S3*.

1.3.4 Amazon DynamoDB

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

After you set up the SDK, you can start using DynamoDB by following the instructions at *Store and Retrieve App Data in Amazon DynamoDB*.

1.3.5 Amazon Kinesis

Amazon Kinesis is a fully managed service for real-time processing of streaming data at massive scale.

After you set up the SDK, you can start using Amazon Kinesis by following the instructions at *Process Streaming Data with Amazon Kinesis and Firehose*.

1.3.6 AWS Lambda

AWS Lambda is a compute service that runs your code in response to requests or events and automatically manages the compute resources for you, making it easy to build applications that respond quickly to new information.

After you set up the SDK, you can start using Lambda by following the instructions at *Execute Code On Demand with Amazon Lambda*.

1.4 What's included in the AWS Mobile SDK for Android?

The AWS SDK for Android includes the following:

- *Class libraries* – Classes that hide much of the lower-level plumbing of the web service interface, including authentication, request retries, and error handling. Each service has its own library, so you can include class libraries for only the services you need and keep your application as small as possible.
- *Code samples* – Practical examples of using the class libraries to build applications.
- *Documentation* – Reference documentation for the AWS SDK for Android.

The SDK is distributed as a `.zip` file containing the following assets:

- `License.txt`
- `Notice.txt`
- `Readme.txt`
- lib/ – Contains Java archive files (`.jar`) that include AWS class libraries. To manage the size of your application, you can include only the files that you need for the services your application is using.
- documentation/ – Includes Javadoc files and other documentation for using the AWS Mobile SDK for Android.
- samples/ – Contains an HTML document with links to samples on GitHub. Samples are named based on the services they demonstrate.
- src/ – Contains an HTML document with links to source on GitHub. Contains the original source files for the class libraries.
- third-party/ – Contains third-party libraries that the SDK depends on.

Important: The AWS Mobile SDK for Android no longer includes a separate JAR for AWS Security Token Service. AWS STS is now bundled with the core, and including AWS STS as a separate JAR will result in a compile-time error.

CHAPTER 2

Set Up the AWS Mobile SDK for Android

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.

2.1 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.

2.2 Step 1: Get the AWS Mobile SDK for Android

There are three ways to get the AWS Mobile SDK for Android.

2.2.1 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 {
    compile 'com.amazonaws:aws-android-sdk-core:2.2.+'
    compile 'com.amazonaws:aws-android-sdk-s3:2.2.+'
    compile 'com.amazonaws:aws-android-sdk-ddb:2.2.+'
}
```

A full list of dependencies are listed below.

Dependency	Build.gradle Value
AWS Mobile SDK core	com.amazonaws:aws-android-sdk-core:2.2.+
Amazon API Gateway	com.amazonaws:aws-android-sdk-apigateway-core:2.2.+
Auto Scaling	com.amazonaws:aws-android-sdk-autoscaling:2.2.+
Amazon Cloud Watch	com.amazonaws:aws-android-sdk-cloudwatch:2.2.+
Amazon Cognito Sync	com.amazonaws:aws-android-sdk-cognito:2.2.+
Amazon Cognito Identity Provider	com.amazonaws:aws-android-sdk-cognitoidentityprovider:2.2.+
Amazon DynamoDB	com.amazonaws:aws-android-sdk-ddb:2.2.+
Amazon DynamoDB Object Mapper	com.amazonaws:aws-android-sdk-ddb-mapper:2.2.+
Amazon EC2	com.amazonaws:aws-android-sdk-ec2:2.2.+
Elastic Load Balancing	com.amazonaws:aws-android-sdk-elb:2.2.+
AWS IoT	com.amazonaws:aws-android-sdk-iot:2.2.+
Amazon Kinesis	com.amazonaws:aws-android-sdk-kinesis:2.2.+
AWS Key Management Service (KMS)	com.amazonaws:aws-android-sdk-kms:2.2.+
Amazon Lex	com.amazonaws:aws-android-sdk-lex:2.3.4@aar
AWS Lambda	com.amazonaws:aws-android-sdk-lambda:2.2.+
Amazon Machine Learning	com.amazonaws:aws-android-sdk-machinelearning:2.2.+
Amazon Mobile Analytics	com.amazonaws:aws-android-sdk-mobileanalytics:2.2.+
Amazon Pinpoint	com.amazonaws:aws-android-sdk-pinpoint:2.3.5
Amazon Polly	com.amazonaws:aws-android-sdk-polly:2.3.4
Amazon S3	com.amazonaws:aws-android-sdk-s3:2.2.+
Amazon Simple DB	com.amazonaws:aws-android-sdk-sdb:2.2.+
Amazon SES	com.amazonaws:aws-android-sdk-ses:2.2.+
Amazon SNS	com.amazonaws:aws-android-sdk-sns:2.2.+
Amazon SQS	com.amazonaws:aws-android-sdk-sqs:2.2.+

2.2.2 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.

2.2.3 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
AWS Mobile SDK Core [1]	aws-android-sdk-core
Amazon API Gateway	aws-android-sdk-apigateway-core
Auto Scaling	aws-android-sdk-autoscaling
Amazon Cloud Watch	aws-android-sdk-cloudwatch
Amazon Cognito Sync	aws-android-sdk-cognito
Amazon Cognito Identity Provider	aws-android-sdk-cognitoidentityprovider
Amazon DynamoDB	aws-android-sdk-ddb
Amazon DynamoDB Object Mapper	aws-android-sdk-ddb-mapper
Amazon EC2	aws-android-sdk-ec2
Elastic Load Balancing	aws-android-sdk-elb
AWS IoT	aws-android-sdk-iot
Amazon Kinesis	aws-android-sdk-kinesis
AWS Key Management Service (KMS)	aws-android-sdk-kms
AWS Lambda	aws-android-sdk-lambda
Amazon Lex	aws-android-sdk-lex
Amazon Machine Learning	aws-android-sdk-machinelearning
Amazon Mobile Analytics	aws-android-sdk-mobileanalytics
Amazon Pinpoint	aws-android-sdk-pinpoint
Amazon Polly	aws-android-sdk-polly
Amazon S3	aws-android-sdk-s3
Amazon Simple DB	aws-android-sdk-sdb
Amazon SES	aws-android-sdk-ses
Amazon SNS	aws-android-sdk-sns
Amazon SQS	aws-android-sdk-sqs

2.3 Step 2: Set Permissions in Your Manifest

Add the following permission to your `AndroidManifest.xml`

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

2.4 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

[1] AWS Mobile SDK Core includes Amazon Cognito Identity and AWS Simple Token Service (STS).

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. These default roles provide your identity pool access to Cognito Sync and Mobile Analytics.

The next page displays code that creates a credentials provider so you can easily integrate Cognito Identity with your 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*/
);
```

2.5 Next Steps

- **Get Started**: View one of our step-by-step Getting Started Guides.
- **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. These are accompanied by detailed integration guidance for your mobile app.
- **Ask questions**: Post questions on the AWS Mobile SDK Forums.

CHAPTER 3

Getting Started with the AWS Mobile SDK for Android

The AWS SDK for Android provides the libraries, samples and, documentation needed to call AWS services from Android apps. This guide will walk you through the following:

3.1 Store and Retrieve Files with Amazon S3 Transfer Utility

Amazon Simple Storage Service (Amazon S3) provides mobile developers with secure, durable, highly-scalable object storage. Amazon S3 is easy to use, with a simple web services interface to store and retrieve any amount of data from anywhere on the web.

The AWS Mobile SDK allows you to consume the S3 service in your mobile application via the S3 Transfer Utility, which is replacing the S3 Transfer Manager as of AWS Mobile SDK for Android v 2.2.4. For information on migrating from the S3 Transfer Manger to the S3 Transfer Utility, see Migrating from the Transfer Manager to the Transfer Utility on the AWS Blog.

The tutorial below explains how to integrate the S3 TransferUtility, a high-level utility for using S3 with your app. This tutorial assumes you have already created an S3 bucket. To create an S3 bucket, visit the S3 AWS Console.

3.1.1 Project Setup

Prerequisites

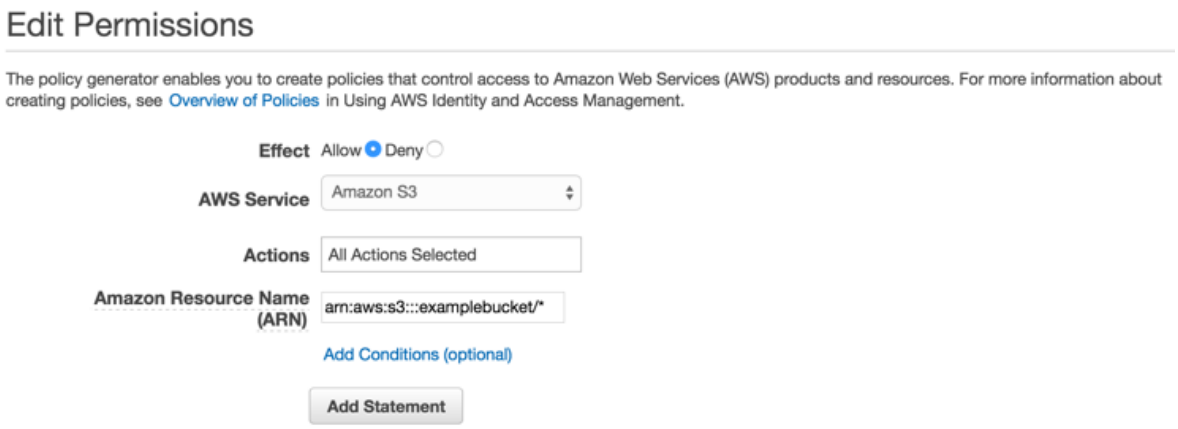
You must complete all of the instructions on the Set Up the SDK for Android page before beginning this tutorial.

Grant Access to Your S3 Resources

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

1. Navigate to the Identity and Access Management Console and click *Roles* in the left-hand pane.

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



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

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

Declare the Service in AndroidManifest.xml

Add the following declaration to your `AndroidManifest.xml`:

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

3.1.2 Initialize the S3 TransferUtility

First, pass your Amazon Cognito credentials provider to the S3 client constructor. Then, pass the client to the TransferUtility constructor along with the application context:

```
AmazonS3 s3 = new AmazonS3Client(credentialsProvider);
TransferUtility transferUtility = new TransferUtility(s3,␣
↪getApplicationContext());
```

3.1.3 Upload a File to Amazon S3

To upload a file to S3, instantiate a `TransferObserver` object. Call `upload()` on your `TransferUtility` object and assign it to the observer, passing the following parameters:

- `bucket_name` – Name of the S3 bucket to store the file
- `key` – Name of the file, once stored in S3
- `file` – java.io.File object to upload

```
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 */
);
```

Uploads automatically use S3's multi-part upload functionality on large files to enhance throughput.

3.1.4 Download a File from Amazon S3

To download a file from S3, instantiate a `TransferObserver` object. Call `download()` on your `TransferUtility` object and assign it to the observer, passing the following parameters:

- `bucket_name` – A string representing the name of the S3 bucket where the file is stored
- `key` – A string representing the name of the S3 object (a file in this case) to download
- `file` – the java.io.File object where the downloaded file will be written

```
TransferObserver observer = transferUtility.download(
  MY_BUCKET,     /* The bucket to download from */
  OBJECT_KEY,    /* The key for the object to download */
  MY_FILE        /* The file to download the object to */
);
```

For more information about accessing Amazon S3 from an Android application, see *Store and Retrieve Files with Amazon S3*.

3.2 Sync User Data with Cognito Sync

Amazon Cognito Sync makes it easy to save mobile user data, such as app preferences or game state in the AWS Cloud without writing any backend code or managing any infrastructure. You can save data locally on users' devices allowing your applications to work even when the devices are offline. You can also synchronize data across a user's devices so that their app experience will be consistent regardless of the device they use.

The tutorial below explains how to integrate Sync with your app.

3.2.1 Project Setup

Prerequisites

You must complete all of the instructions on the Set Up the SDK for Android page before beginning this tutorial.

3.2.2 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 *Authenticate Users with Amazon Cognito Identity*.

3.2.3 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 `Dataset`. User data is added in the form of key/value pairs. Dataset objects are created with the `CognitoSyncManager` class which functions as a Cognito client object. Use the defaultCognito method to get a reference to the instance of CognitoSyncManager. The openOrCreateDataset method is used to create a new dataset or open an existing instance of a dataset stored locally on the device:

```
Dataset dataset = client.openOrCreateDataset("datasetname");
```

Cognito datasets function as dictionaries, with values accessible by key:

```
String value = dataset.get("myKey");
dataset.put("myKey", "my value");
```

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.

3.3 Store and Query App Data in DynamoDB

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

The tutorial below explains how to integrate the DynamoDB ObjectMapper with your app, which stores Java objects in DynamoDB.

3.3.1 Project Setup

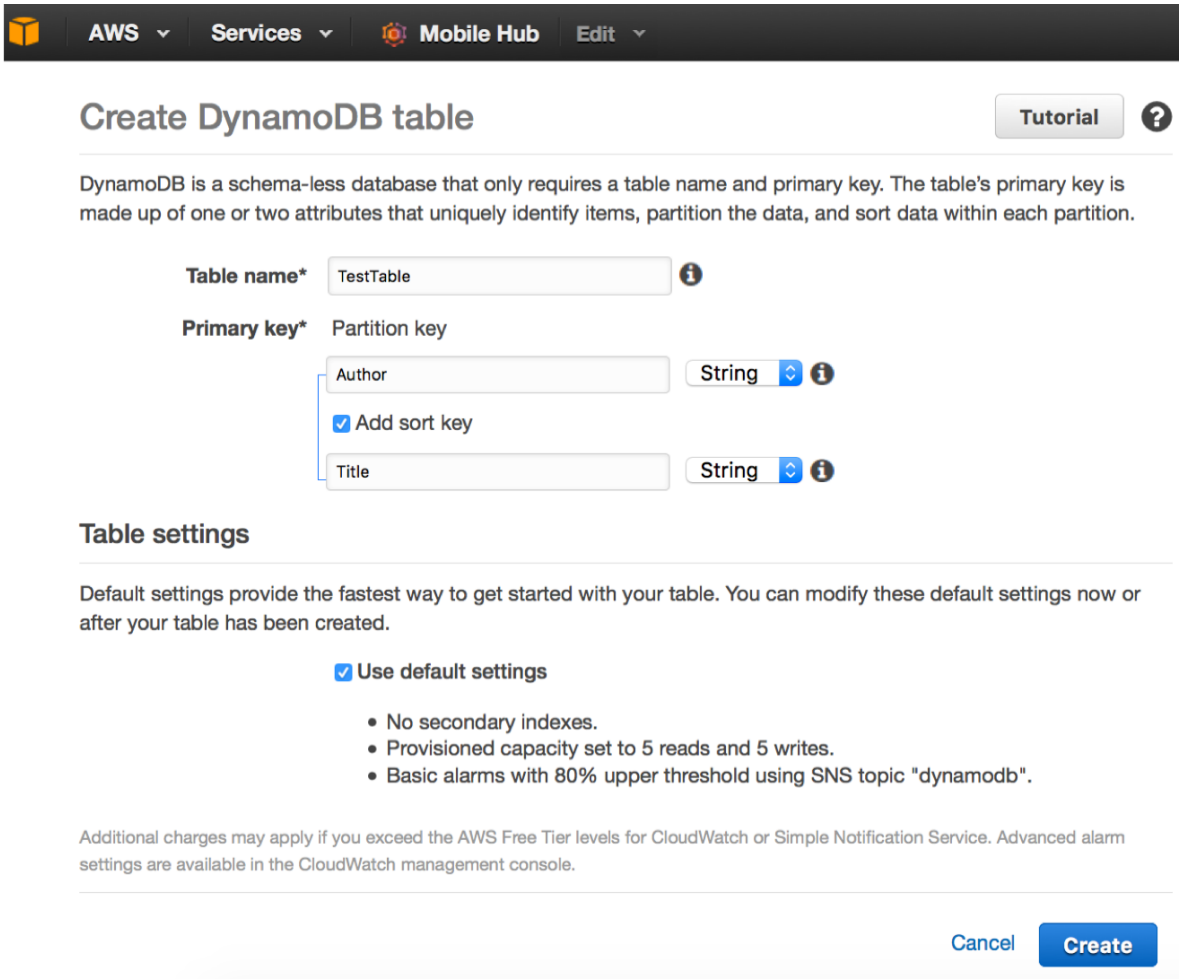
Prerequisites

You must complete all of the instructions on the Set Up the SDK for Android page before beginning this tutorial.

3.3.2 Create a DynamoDB Table

Before you can read and write data to a DynamoDB database, you must create a table. When creating a table you must specify the primary key. The primary key is composed of a partition key attribute and an optional sort key attribute. For more information on how primary and sort attributes are used, see Working With Tables.

1. To invoke the wizard, navigate to the DynamoDB Console and choose *Create Table*.
2. For *Table name* type *Books*.
3. In *Primary key*, for *Partition key*, type *Author* for the partition key value and choose *String* for the key type.
4. Choose *Add sort key*.
5. In *Add sort key*, type *Title* for the sort key value and choose *String* for the key type.



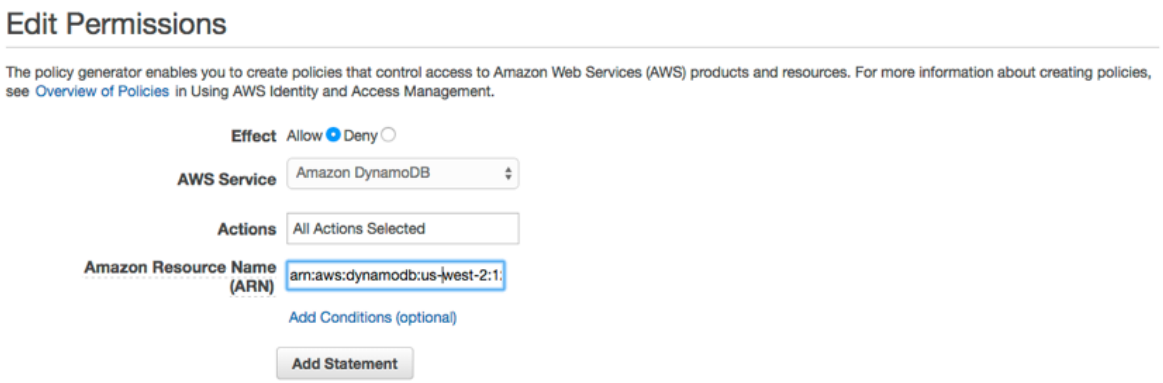
6. Keep *Use default settings* selected and click *Create*.

3.3.3 Update IAM Roles

In order for your Cognito identity pool to access Amazon DynamoDB, you must modify the identity pool's roles.

1. Navigate to the Identity and Access Management Console and click *Roles* in the left-hand pane and search for your Identity Pool name - two roles will be listed one for unauthenticated users and one for authenticated users.
2. Click the role for unauthenticated users (it will have "unauth" appended to your Identity Pool name) and click the *Create Role Policy* button.
3. Select *Policy Generator* and click the *Select* button.
4. In the Edit Permissions page enter the settings shown in the following image. The Amazon Resource Name (ARN) of a DynamoDB table looks like `arn:aws:dynamodb:us-west-2:123456789012:table/my-table-name` and is composed of the region in which the table is located, the owner's AWS account number, and the name of the table in the format `table/my-table-name`. For more information about specifying

ARNs, see Amazon Resource Names for DynamoDB.



5. Click the *Add Statement* button, click the *Next Step* button and the Wizard will show you the configuration generated.
6. Click the *Apply Policy* button.

Add Import Statements

Add the following imports to the main activity of your app:

```
import com.amazonaws.auth.CognitoCachingCredentialsProvider;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.dynamodbv2.*;
import com.amazonaws.mobileconnectors.dynamodbv2.dynamodbmapper.*;
```

3.3.4 Initialize AmazonDynamoDBClient

Pass your initialized Amazon Cognito credentials provider to the `AmazonDynamoDB` constructor:

```
AmazonDynamoDBClient ddbClient = new␣
↪AmazonDynamoDBClient(credentialsProvider);
```

3.3.5 Initialize DynamoDBMapper

Pass your initialized DynamoDB client to the `DynamoDBMapper` constructor:

```
DynamoDBMapper mapper = new DynamoDBMapper(ddbClient);
```

3.3.6 Write a Row

To write a row to the table, define a class to hold your row data. This class must be derived from `AWSDynamoDBModel` and implement the `AWSDynamoDBModel` interface. The class should also

contain properties that hold the attribute data for the row. The following class declaration illustrates such a class:

```
@DynamoDBTable(tableName = "Books")
    public class Book {
        private String title;
        private String author;
        private int price;
        private String isbn;
        private Boolean hardCover;

        @DynamoDBIndexRangeKey(attributeName = "Title")
        public String getTitle() {
            return title;
        }

        public void setTitle(String title) {
            this.title = title;
        }

        @DynamoDBIndexHashKey(attributeName = "Author")
        public String getAuthor() {
            return author;
        }

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

        @DynamoDBAttribute(attributeName = "Price")
        public int getPrice() {
            return price;
        }

        public void setPrice(int price) {
            this.price = price;
        }

        @DynamoDBHashKey(attributeName = "ISBN")
        public String getIsbn() {
            return isbn;
        }

        public void setIsbn(String isbn) {
            this.isbn = isbn;
        }

        @DynamoDBAttribute(attributeName = "Hardcover")
        public Boolean getHardCover() {
            return hardCover;
        }

        public void setHardCover(Boolean hardCover) {
            this.hardCover = hardCover;
```

```
        }
    }
```

To save an object, first create it and set the appropriate fields:

```
Book book = new Book();
book.setTitle("Great Expectations");
book.setAuthor("Charles Dickens");
book.setPrice(1299);
book.setIsbn("1234567890");
book.setHardCover(false);
```

Then save the object:

```
mapper.save(book);
```

To update a row, modify the instance of the `DDTableRow` class and call `AWSDynamoObjectMapper.save()` as shown above.

3.3.7 Retrieve a Row

Retrieve an object using a primary key:

```
Book selectedBook = mapper.load(Book.class, "1234567890");
```

For more information on accessing DynamoDB from an Android application, see Amazon Dynamo DB.

3.4 Track App Usage Data with Amazon Mobile Analytics

Amazon Mobile Analytics allows you to measure app usage and app revenue. By tracking key trends such as new vs. returning users, app revenue, user retention, and custom in-app behavior events, you can make data-driven decisions to increase engagement and monetization for your app.

The tutorial below explains how to integrate Mobile Analytics with your app.

3.4.1 Project Setup

Prerequisites

You must complete all of the instructions on the *Set Up the AWS Mobile SDK for Android* page before beginning this tutorial.

Create an App in the Mobile Analytics Console

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

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

Set Permissions in Your Android Manifest

In `AndroidManifest.xml`, set the following permissions:

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

3.4.2 Initialize MobileAnalyticsManager

Define a static reference to the `MobileAnalyticsManager` in the `onCreate()` method of your main activity:

```
private static MobileAnalyticsManager analytics;
```

For this particular example, let's also create two constants that we'll use later in a custom event:

```
private static final int STATE_LOSE = 0;
private static final int STATE_WIN = 1;
```

In the activity's onCreate() method, create an instance of MobileAnalyticsManager. You'll need to replace "cognitoId" and "appId" to their respective values as shown from the Mobile Analytics console. The appId is used to group your data in the console.

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

    try {
        analytics = MobileAnalyticsManager.getOrCreateInstance(
                    this.getApplicationContext(),
                    "appId",
                    "identityPoolId"
        );
    } catch(InitializationException ex) {
            Log.e(this.getClass().getName(), "Failed to initialize Amazon␣
↪Mobile Analytics", ex);
    }
}
```

By default, the MobileAnalyticsManager client initializes with WAN delivery enabled.

3.4.3 Track Session Events

Override the activity's `onPause()` and `onResume()` methods to record session events:

```
/**
 * Invoked when the Activity loses user focus.
 */
@Override
protected void onPause() {
    super.onPause();
    if(analytics != null) {
        analytics.getSessionClient().pauseSession();
        //Attempt to send any events that have been recorded to the Mobile␣
↪Analytics service
        analytics.getEventClient().submitEvents();
    }
}

@Override
protected void onResume() {
    super.onResume();
    if(analytics != null)  {
        analytics.getSessionClient().resumeSession();
    }
}
```

For each activity in your application, you will need to record session events in the `onPause()` and `onResume()` methods.

Add Monetization Events

The AWS Mobile SDK for Android provides a `MonetizationEventBuilder` that lets you create events for Amazon purchases, Google Play purchases, and virtual store purchases. The `MonetizationEventBuilder` class can be extended if you need to record monetization events from other purchase frameworks.

To learn more about adding monetization events, see the API reference guide for MonetizationEventBuilder.

Record Custom Events

The Mobile Analytics client lets you create and record custom events. For example, if our app were a game, we might create a custom event to be submitted when the user completes a level. In your main activity, add the following method, which creates and records a custom event.

```
/**
* This method gets called when the player completes a level
* @param levelName the name of the level
* @param difficulty the difficulty setting
* @param timeToComplete the time to complete the level in seconds
* @param playerState the winning/losing state of the player
*/
public void onLevelComplete(String levelName, String difficulty, double␣
↪timeToComplete, int playerState) {
```

```
    //Create a Level Complete event with some attributes and␣
↪metrics(measurements)
    //Attributes and metrics can be added using with statements
    AnalyticsEvent levelCompleteEvent = analytics.getEventClient().
↪createEvent("LevelComplete")
        .withAttribute("LevelName", levelName)
        .withAttribute("Difficulty", difficulty)
        .withMetric("TimeToComplete", timeToComplete);

    //attributes and metrics can also be added using add statements
    if (playerState == STATE_LOSE)
        levelCompleteEvent.addAttribute("EndState", "Lose");
    else if (playerState == STATE_WIN)
        levelCompleteEvent.addAttribute("EndState", "Win");

    //Record the Level Complete event
    analytics.getEventClient().recordEvent(levelCompleteEvent);
}
```

3.5 Writing App Data to Amazon Kinesis

Amazon Kinesis is a fully managed service for real-time processing of streaming data at massive scale. 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. The tutorial below explains how to integrate Amazon Kinesis and/or Amazon Kinesis Firehose with your app.

3.5.1 Project Setup

Prerequisites

You must complete all of the instructions on the Set Up the SDK for Android page before beginning this tutorial.

3.5.2 Initialize KinesisRecorder for Amazon Kinesis

Pass your initialized Amazon Cognito credentials provider to the `KinesisRecorder` constructor:

```
String kinesisDirectory = "YOUR_UNIQUE_DIRECTORY";
KinesisRecorder recorder = new KinesisRecorder(
        myActivity.getDir(kinesisDirectory, 0), // An empty directory␣
↪KinesisRecorder can use for storing requests
        Regions.US_WEST_2,  // Region that this Recorder should save and send␣
↪requests to
        credentialsProvider); // The credentials provider to use when making␣
↪requests to AWS
```

3.5.3 Initialize KinesisFirehoseRecorder for Amazon Kinesis Firehose

Pass your initialized Amazon Cognito credentials provider to the `KinesisFirehoseRecorder` constructor:

```
String kinesisFirehoseDirectory = "YOUR_UNIQUE_DIRECTORY";
KinesisFirehoseRecorder recorder = new KinesisFirehoseRecorder(
        myActivity.getDir(kinesisFirehoseDirectory, 0), /* An empty directory␣
↪KinesisFirehoseRecorder can use for storing requests */
        Regions.US_WEST_2,  /* Region that this Recorder should save and send␣
↪requests to */
        credentialsProvider); /* The credentials provider to use when making␣
↪requests to AWS */
```

3.5.4 Create a Kinesis Stream

In order to use the Amazon Kinesis recorder, you must first create an Amazon Kinesis stream. You can create new streams in the Kinesis Console.

3.5.5 Grant Role Access to Your Amazon Kinesis Stream

The default IAM role policy grants you access to Amazon Mobile Analytics and Amazon Cognito Sync. To use Amazon Kinesis in an application, you must allow the IAM roles associated with your Cognito Identity Pool access to your Kinesis stream. To set this policy:

1. Navigate to the Identity and Access Management Console and choose *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. Choose the role for unauthenticated users (it will have "unauth" appended to your Identity Pool name).
4. Scroll down the web page until you see the *Create Role Policy*. Choose it, select *Policy Generator*, and then choose the *Select* button.
5. Select the *Allow* radio button, Amazon Kinesis in the *AWS Service* drop-down, PutRecord under *Actions*, and enter the ARN to your Kinesis stream in the *Amazon Resource Name (ARN)* text box.

   ```
   "Resource": arn:aws:kinesis:region:account:stream/name
   "Resource": arn:aws:kinesis:us-west-2:111122223333:stream/my-stream
   ```

6. Choose the *Add Statement* button, the *Next Step* button, and the *Apply Policy* button.

To learn more about Kinesis-specific policies, see Controlling Access to Amazon Kinesis Resources with IAM.

3.5.6 Grant Role Access to Your Kinesis Firehose Delivery Stream

The default IAM role policy grants you access to Amazon Mobile Analytics and Amazon Cognito Sync. To use Kinesis Firehose in an application, you must allow the IAM roles associated with your Amazon Cognito Identity Pool access to your Kinesis Firehose delivery stream. To set this policy:

1. Navigate to the Identity and Access Management Console and choose *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. Choose the role for unauthenticated users (it will have "unauth" appended to your Identity Pool name).
4. Scroll down the web page until you see the *Create Role Policy*. Choose it, select *Policy Generator*, and then choose the *Select* button.
5. Select the *Allow* radio button, Amazon Kinesis in the *AWS Service* drop-down, PutRecord under *Actions*, and enter the ARN to your Kinesis stream in the *Amazon Resource Name (ARN)* text box.

```
"Resource": arn:aws:firehose:region:account:stream/name
"Resource": arn:aws:firehose:us-west-2:111122223333:deliverystream/my-
↪stream
```

6. Choose the *Add Statement* button, the *Next Step* button, and the *Apply Policy* button.

To learn more about Kinesis Firehose-specific policies, see Controlling Access to Amazon Kinesis Firehose.

3.5.7 Configure the Kinesis Service Client

Use the `KinesisRecorder` class to interact with the Kinesis service. The following snippet creates an instance of the Kinesis service client:

```
String kinesisDirectory = "YOUR_UNIQUE_DIRECTORY";
KinesisRecorder recorder = new KinesisRecorder(
        myActivity.getDir(kinesisDirectory, 0),
        Regions.US_WEST_2,
        credentialsProvider);
```

`YOUR_UNIQUE_DIRECTORY` is a folder that should be exclusive to the Kinesis Recorder and will be used to store records. The region here should match the region you specified in the console.

Note: `KinesisRecorder` uses synchronous calls, so you shouldn't call `KinesisRecorder` methods on the main thread.

3.5.8 Save Records to Local Storage

With `KinesisRecorder` created and configured, you can use `saveRecord()` to save records to local storage:

```
recorder.saveRecord("MyData".getBytes(),"MyStreamName");
```

3.5.9 Submit Records to Kinesis Stream

Use the `submitAllRecords` synchronous method on the `KinesisRecorder` object to send all locally saved records to your Kinesis stream.

```
recorder.submitAllRecords();
```

To learn more about working with Amazon Kinesis, see the Amazon Kinesis Developer Resources.

To learn more about working with Amazon Kinesis Firehose, see the Amazon Kinesis Firehose Documentation.

To learn more about the Kinesis classes, see the class reference for AWSKinesisRecorder.

3.6 Mobile Backend Using Amazon Lambda

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.

3.6.1 Project Setup

Prerequisites

You must complete all of the instructions on the Set Up the SDK for Android 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 Amazon 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",
      ]
   }]
}
```

6. Click the *Add Statement* button, and then click the *Next Step* button. The wizard will show you the configuration that you generated.
7. 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" />
```

3.6.2 Initialize LambdaInvokerFactory

Pass your initialized Amazon Cognito credentials provider to the `LambdaInvokerFactory` constructor:

```
LambdaInvokerFactory factory = new LambdaInvokerFactory(
  myActivity.getApplicationContext(),
  REGION,
  credentialsProvider);
```

3.6.3 Declare Data Types

Declare the Java classes to hold the data you pass to the Lambda function. The following class defines a NameInfo class that contains a person's first and last name:

```
package com.amazonaws.demo.lambdainvoker;

/**
 * A simple POJO
 */
 public class NameInfo {
    private String firstName;
    private String lastName;

    public NameInfo() {}

    public NameInfo(String firstName, String lastName) {
        this.firstName = firstName;
        this.lastName = lastName;
    }

    public String getFirstName() {
       return firstName;
    }

    public void setFirstName(String firstName) {
       this.firstName = firstName;
    }

    public String getLastName() {
       return lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }
}
```

3.6.4 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:

```
package com.amazonaws.demo.lambdainvoker;
import com.amazonaws.mobileconnectors.lambdainvoker.LambdaFunction;

/*
 * A holder for lambda functions
 */
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);
}
```

3.6.5 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.

```
// 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);

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);
```

Now whenever the Lambda function is invoked, you should see an application toast with the text “Hello John using <device>”.

For more information on accessing AWS Lambda, see *Execute Code On Demand with Amazon Lambda*.

3.7 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.

3.7.1 Project 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

The default IAM role policy grants you access to Amazon Mobile Analytics and Amazon Cognito Sync. 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.*;
```

3.7.2 Initialize AmazonMachineLearningClient

Pass your initialized Amazon Cognito credentials provider to the `AmazonMachineLearningClient` constructor:

```
AmazonMachineLearningClient client = new␣
↪AmazonMachineLearningClient(credentialsProvider);
```

3.7.3 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:

```
// 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;
}
```

Once the real-time endpoint is ready, we can begin calling Predict. Note that you must pass the real-time endpoint through the PredictRequest.

```
// 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);
System.out.println(predictResult.getPrediction());

// Do something with the prediction
// ...
```

Additional Resources

- Developer Guide
- Service API Reference

3.8 Adding Natural Language Speech and Text to your App

3.8.1 What is Amazon Lex?

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.

3.8.2 Setting Up

Include the SDK in Your Project

Follow the instructions at *Set Up the AWS Mobile SDK for Android* 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.

```
compile '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

To setup interaction between your mobile app and Amazon Lex, use the Amazon Lex console to configure a bot that fulfills your requirements. To learn more see *Lex Developer Guide*_. For a quickstart, see Step 2 of 'Getting Started <http://docs.aws.amazon.com/lex/latest/dg/gs-bp-prep.html>'_ in the *Lex Developer Guide*.

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.

3.8.3 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.

```
CognitoCredentialsProvider credentialsProvider = new␣
↪CognitoCredentialsProvider(
            appContext.getResources().getString(R.string.identity_id_
↪test),
            Regions.fromName(appContext.getResources().getString(R.
↪string.aws_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

```
// 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);
```

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:

```
lexInteractionClient.textInForTextOut(String text, Map<String,␣
↪String> sessionAttributes)
```

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 relatedto 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`.

```
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>));
}
```

3.9 Adding Text to Speech Conversion to your App

3.9.1 What is Amazon Polly?

Amazon Polly is an AWS service for converting text to Speech.

The AWS Mobile SDK for Android provides a simple client for generating speech audio playback from text provided to the Polly service by your app.

For information on Amazon Polly concepts and service configuration, see How it Works in the *Amazon Polly Developer Guide*.

3.9.2 Implement Text to Speech Conversion with Amazon Polly

See information on Amazon Polly implementation for Android apps in the Application Examples section of the *Polly Developer Guide*.

3.10 Create Push Notification Campaigns with Amazon Pinpoint

3.10.1 What is Amazon Pinpoint?

Using Amazon Pinpoint, you can create push notification campaigns that provide your users with timely, relevant, personalized information to encourage them to keep using your mobile app. You can use push notification campaigns to increase app awareness, downloads, and launches; build customer loyalty; and ultimately boost your mobile revenues.

Campaigns can include things such as welcome and onboarding messages, invites to less-engaged or lapsed users, location-based invites, time-based reminders, and special offers for frequent users. You can use Amazon Pinpoint to create and push different campaigns based on groupings of specific user characteristics, which we refer to as user segments.

Using the following resources, you can integrate Amazon Pinpoint with your Android app to make it ready to be part of a campaign.

For information on Amazon Pinpoint concepts and service configuration, see the Amazon Pinpoint Developer Guide.

For end to end sample apps using Amazon Pinpoint see the AWS SDK for Android samples.

3.10.2 Make your app ready to be in an Amazon Pinpoint campaign

See information on Amazon Pinpoint implementation for Android apps in the Integrating Amazon Pinpoint With Android Apps section of the Amazon Pinpoint Developer Guide.

For information about other AWS Mobile SDKs, see AWS Mobile SDK.

CHAPTER 4

Authenticate Users with Amazon Cognito Identity

4.1 What is Amazon Cognito Identity?

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

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

For information about Cognito Authentication Region availability, see AWS Service Region Availability.

4.2 Using a Public Provider to Authenticate Users

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

4.3 Using Developer Authenticated Identities

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

CHAPTER 5

Sync User Data with Amazon Cognito Sync

5.1 What is Amazon Cognito Sync?

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.

CHAPTER 6

Track App Usage Data with Amazon Mobile Analytics

6.1 What is Amazon Mobile Analytics?

Amazon Mobile Analytics lets you collect, visualize, and understand app usage for your Android and Fire OS apps. Reports are available for metrics on active users, sessions, retention, in-app revenue, and custom events, and can be filtered by platform and date range. Amazon Mobile Analytics is built to scale with your business and can collect and process billions of events from many millions of endpoints.

Using Amazon Mobile Analytics, you can track customer behaviors, aggregate metrics, generate data visualizations, and identify meaningful patterns. The AWS SDK for Android provides integration with the Mobile Analytics service.

For information about Mobile Analytics Region availability, see AWS Service Region Availability.

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

6.2 Getting Started

6.2.1 Create an App in the Mobile Analytics Console

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

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

6.2.2 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 Cognito Identity in your Android application.

For more information on Cognito Identity, see *Authenticate Users with Amazon Cognito Identity*.

6.2.3 IAM Policy for Amazon Mobile Analytics

To use Mobile Analytics, AWS users must have the correct permissions. The following IAM policy allows the user to submit events to Mobile Analytics:

```
{
    "Statement": [{
        "Effect": "Allow",
        "Action": "mobileanalytics:PutEvents",
        "Resource": "*"
    }]
}
```

This policy should be assigned to roles associated with the Cognito identity pool for your app. The policy allows clients to record events with the Mobile Analytics service. Amazon Cognito will set this policy for you, if you let it create new roles. Other policies are required to allow IAM users to view reports.

You can set permissions at the IAM Console. To learn more about IAM policies, see Using IAM.

6.2.4 Include the SDK in Your Project

Follow the instructions on the *Set Up the AWS Mobile SDK for Android* page to include the proper JAR files for this service and set the appropriate permissions.

6.2.5 Set Permissions in Your Android Manifest

In `AndroidManifest.xml`, set the following permissions, if they're not already present:

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

6.2.6 Initialize MobileAnalyticsManager

Define a static reference to the `MobileAnalyticsManager` in the `onCreate()` method of your main activity:

```
private static MobileAnalyticsManager analytics;
```

For this particular example, let's also create two constants that we'll use later in a custom event:

```
private static final int STATE_LOSE = 0;
private static final int STATE_WIN = 1;
```

In the activity's onCreate() method, create an instance of MobileAnalyticsManager. You'll need to replace "cognitoId" and "appId" to their respective values as shown from the Mobile Analytics console. The appId is used to group your data in the Mobile Analytics console.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    try {
        analytics = MobileAnalyticsManager.getOrCreateInstance(
                    this.getApplicationContext(),
                    "appId",
                    "identityPoolId"
        );
    } catch(InitializationException ex) {
            Log.e(this.getClass().getName(), "Failed to initialize Amazon␣
↪Mobile Analytics", ex);
    }
}
```

By default, the MobileAnalyticsManager client initializes with WAN delivery enabled.

6.2.7 Track Session Events

Override the activity's `onPause()` and `onResume()` methods to record session events.

```
/**
 * Invoked when the Activity loses user focus
 */
@Override
protected void onPause() {
```

```
    super.onPause();
    if(analytics != null) {
        analytics.getSessionClient().pauseSession();
        //Attempt to send any events that have been recorded to the Mobile␣
↪Analytics service.
        analytics.getEventClient().submitEvents();
    }
}

@Override
protected void onResume() {
    super.onResume();
    if(analytics != null)  {
        analytics.getSessionClient().resumeSession();
    }
}
```

For each activity in your application, you will need to record session events in the `onPause()` and `onResume()` methods.

Add Monetization Events

The SDK for Android provides a `MonetizationEventBuilder` that lets you create events for Amazon purchases, Google Play purchases, and virtual store purchases. The `MonetizationEventBuilder` class can be extended if you need to record monetization events from other purchase frameworks.

To learn more about adding monetization events, see the API reference guide for MonetizationEventBuilder.

Record Custom Events

The Mobile Analytics client lets you create and record custom events. For example, if our app were a game, we might create a custom event to be submitted when the user completes a level. In your main activity, add the following method, which creates and records a custom event.

```
/**
* This method gets called when the player completes a level
* @param levelName the name of the level
* @param difficulty the difficulty setting
* @param timeToComplete the time to complete the level in seconds
* @param playerState the winning/losing state of the player
*/
public void onLevelComplete(String levelName, String difficulty, double␣
↪timeToComplete, int playerState) {

    //Create a Level Complete event with some attributes and␣
↪metrics(measurements)
    //Attributes and metrics can be added using with statements
    AnalyticsEvent levelCompleteEvent = analytics.getEventClient().
↪createEvent("LevelComplete")
```

```
            .withAttribute("LevelName", levelName)
            .withAttribute("Difficulty", difficulty)
            .withMetric("TimeToComplete", timeToComplete);

    //attributes and metrics can also be added using add statements
    if (playerState == STATE_LOSE)
        levelCompleteEvent.addAttribute("EndState", "Lose");
    else if (playerState == STATE_WIN)
        levelCompleteEvent.addAttribute("EndState", "Win");

    //Record the Level Complete event
    analytics.getEventClient().recordEvent(levelCompleteEvent);
}
```

Test this custom event by calling it at the end of the `onCreate()` method:

```
this.onLevelComplete("Lower Dungeon", "Very Difficult", 2734, STATE_WIN);
```

Launch and test your app.

Navigate to the Mobile Analytics Console. Your app should appear in the drop-down list, though it may take a few minutes for a new app to appear in the list. You should see session-related data for your app in the graphs.

CHAPTER 7

Store and Retrieve Files with Amazon S3

7.1 What is Amazon S3?

Amazon Simple Storage Service provides secure, durable, highly-scalable object storage in the cloud. Using the AWS Mobile SDK, you can directly access Amazon S3 from your mobile app. For information about S3 Region availability, see AWS Service Region Availability.

The AWS Mobile SDK allows you to consume the S3 service in your mobile application via the S3 Transfer Utility, which is replacing the S3 Transfer Manager as of AWS Mobile SDK for Android v 2.2.4.

For information on migrating from the S3 Transfer Manager to the S3 Transfer Utility, see Migrating from the Transfer Manager to the Transfer Utility on the AWS Blog.

For a complete sample that shows how to use the TransferUtility class to perform download and upload tasks, and manage the tasks, see Running S3TransferUtility Sample

7.2 Setup

7.2.1 Prerequisites

You must complete all of the instructions on the *Set Up the AWS Mobile SDK for Android* page before beginning this tutorial.

7.2.2 Create and Configure an S3 Bucket

Amazon S3 stores your application's resources in buckets—cloud storage containers that live in a specific region.

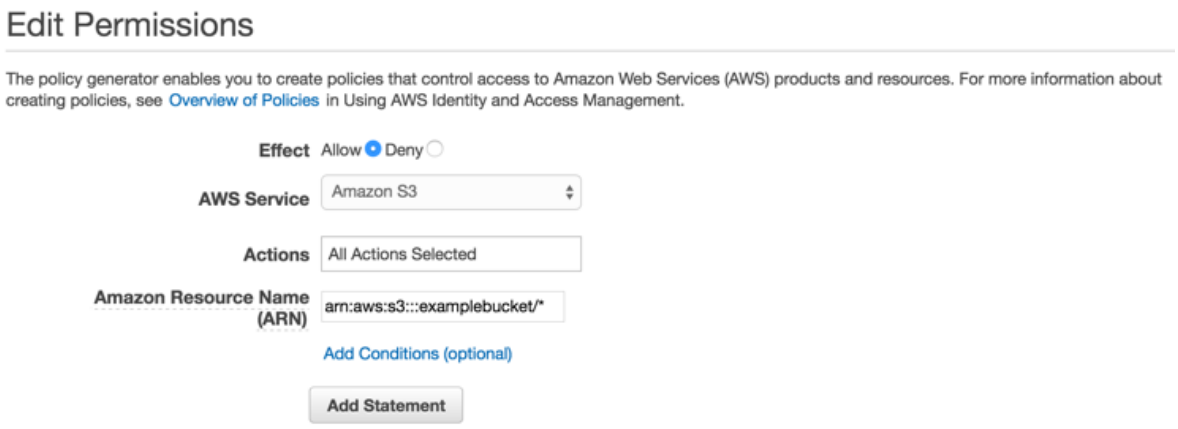
Create a Bucket

1. Sign in to the S3 Console and click *Create Bucket*.
2. Enter a bucket name, select a region, and click *Create*. Each S3 bucket must have a globally unique name.

Grant Access to Your S3 Resources

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

1. Navigate to the Identity and Access Management Console and click *Roles* in the left-hand pane.
2. Type your identity pool name into the search box. Two roles will be listed: one for unauthenticated users and one for authenticated users.
3. Click the role for unauthenticated users (it will have unauth appended to your Identity Pool name).
4. Click the *Create Role Policy* button, select *Policy Generator*, and then click the *Select* button.
5. On the Edit Permissions page, enter the settings shown in the following image. The Amazon Resource Name (ARN) of an S3 bucket looks like `arn:aws:s3:::examplebucket/*` and is composed of the region in which the bucket is located and the name of the bucket. The settings shown below will give your identity pool full to access to all actions for the specified bucket.



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

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

7.2.3 Configure Your Environment

To start using S3 in your application, you need to do the following:

- Add the correct import statements
- Declare the S3 TransferUtility service in your manifest files
- Instantiate a Cognito Caching Credentials Provider, an Amazon S3 client, and a Transfer Utility

Declare the Service in AndroidManifest.xml

Add the following declaration to your `AndroidManifest.xml`:

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

Instantiate an S3 Client

Pass your credentials provider to the S3 client constructor, like so:

```
// Create an S3 client
AmazonS3 s3 = new AmazonS3Client(credentialsProvider);
```

Instantiate TransferUtility

You will use the `TransferUtility` class to upload and download files from S3. Pass the S3 client and the application context to the Transfer Utility, like so:

```
TransferUtility transferUtility = new TransferUtility(s3, APPLICATION_
↪CONTEXT);
```

7.3 Operations

For a complete working sample, see S3 Transfer Utility Sample.

7.3.1 Upload an Object to S3

To upload a file:

```
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 */
);
```

Uploads automatically use S3's multi-part upload functionality on large files to enhance throughput.

7.3.2 Upload an Object to S3 with Metadata

Create a `ObjectMetadata` object:

```
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 low-level S3 `getObjectMetadata` method. For more information, see the API Reference.

7.3.3 Download an Object from S3

To download a file:

```
TransferObserver observer = transferUtility.download(
  MY_BUCKET,     /* The bucket to download from */
  OBJECT_KEY,    /* The key for the object to download */
  MY_FILE        /* The file to download the object to */
);
```

7.3.4 Tracking S3 Transfer Progress

With the Transfer Utility, the `download()` and `upload()` methods return a `TransferObserver` object. This object gives access to:

- The state (now specified as an enum)
- The total bytes transferred thus far
- The total bytes to transfer (for easily calculating progress bars)
- A unique ID that you can use to keep track of distinct transfers

Given the transfer ID, this `TransferObserver` object can be retrieved from anywhere in your app, including if the app is killed. 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 download or upload, call `setTransferListener()` on your `TransferObserver`. This requires you to implement `onStateChanged`, `onProgressChanged`, and `onError`. For example:

```
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
    }

});
```

7.3.5 Pause an S3 Transfer

If an app is killed, crashes, or loses Internet connectivity, transfers are automatically paused. If the device running your app loses network connectivity, paused transfers will automatically resume when the network is available again. If the transfer was manually paused, or the app was killed, transfers can be resumed with the `resume(transferId)` method.

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);
```

You can also query for `TransferObservers`, which contain the transfer ID, with either `getTransfersWithType(transferType)` or `getTransfersWithTypeAndState(transferType,transferState)`. This means that if your app is killed or crashes during a transfer, you can manually determine if there are any paused transfers when the app resumes and handle those as you see fit.

7.3.6 Resume a Transfer

To resume a single transfer:

```
transferUtility.resume(idOfTransferToBeResumed);
```

7.3.7 Cancel a Transfer

Canceling an upload or download is simple. Just call `cancel()` or `cancelAllWithType()` on the Transfer Utility object.

To cancel a single transfer:

```
transferUtility.cancel(idToBeCancelled);
```

To cancel all transfers of a certain type:

```
transferUtility.cancelAllWithType(TransferType.DOWNLOAD);
```

CHAPTER 8

Store and Retrieve App Data in Amazon DynamoDB

8.1 What is Amazon DynamoDB?

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

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

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

8.2 Getting Started

This section provides a step-by-step guide for getting started with DynamoDB using the AWS Mobile SDK for Android. You can also try out the DynamoDB sample.

8.2.1 Include the JAR Files in Your Project

Follow the instructions on the *Set Up the AWS Mobile SDK for Android* page to include the proper JAR files for this service and set the appropriate permissions.

8.2.2 Add Import Statements

Add the following imports to the main activity of your app:

```
import com.amazonaws.auth.CognitoCachingCredentialsProvider;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.dynamodbv2.*;
import com.amazonaws.mobileconnectors.dynamodbv2.dynamodbmapper.*;
import com.amazonaws.services.dynamodbv2.model.*;
```

8.2.3 Set Permissions in Your Android Manifest

In `AndroidManifest.xml`, set the following permission, if it's not already present

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

8.2.4 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 *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 roles with access to your new identity pool.

The next page displays code that creates a credentials provider so you can easily integrate Cognito Identity in your Android application.

For more information about Amazon Cognito Identity, see *Authenticate Users with Amazon Cognito Identity*.

8.2.5 Create a DynamoDB Table

Let's assume we're building a bookstore app. The app will need to keep track of books available in a bookstore, and we can create a DynamoDB table to do so.

To create the Books table:

1. Log in to the DynamoDB Console.
2. Click *Create Table*.
3. Enter **`Books`** as the name of the table.
4. Enter **`ISBN`** in the *Partition key* field of the *Primary key* with *String* as its type.
5. Uncheck the *Use default settings* checkbox and click + *Add Index*.
6. In the *Add Index* dialog enter **`Author`** with *String* as its type.
7. Check the *Add sort key* checkbox and enter **`Title`** as the sort key value, with *String* as its type.

8. Leave the other values at their defaults and click *Add index* to add the **`Author-Title-index`** index.
9. Set the read capacity to `10` and the write capacity to `5`.
10. Click *Create*. DynamoDB will create your database.
11. Refresh the console and select your Books table from the list of tables.
12. Open the *Overview* tab and copy or note the Amazon Resource Name (ARN). You'll need this in a moment.

8.2.6 Set Permissions

To use DynamoDB in an application, you must set the correct permissions. The following IAM policy allows the user to perform the actions shown in this tutorial on two resources (a table and an index) identified by ARN.

```
{
    "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 this policy to the unauthenticated role assigned to your Cognito identity pool, replacing the `Resource` values with the correct ARN for your DynamoDB table:

1. Log in to the IAM Console.
2. Select *Roles* and select the "Unauth" role that Cognito created for you.
3. Click *Attach Role Policy*.
4. Select *Custom Policy* and click *Select*.
5. Enter 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 database; then append `/index/*` to obtain the value for the index ARN.
6. Click *Apply Policy*.

To learn more about IAM policies, see Overview of IAM Policies. To learn more about DynamoDB-specific policies, see *DynamoDB Developer Guide* Authentication and Access Control.

8.3 Create a DynamoDB Client and Object Mapper

We're going to use the DynamoDB Object Mapper to map a client-side class to our database. To use the Object Mapper, we first have to instantiate a DynamoDB client.

When we created an identity pool, we copied the Cognito client initialization code into our app. Assuming that we have a `credentialsProvider` variable holding a reference to our Cognito credential provider, we can create a DynamoDB client as follows:

```
AmazonDynamoDBClient ddbClient = new␣
↪AmazonDynamoDBClient(credentialsProvider);
```

Then we can use our DynamoDB client to create an Object Mapper:

```
DynamoDBMapper mapper = new DynamoDBMapper(ddbClient);
```

Now we're ready to map a class to our database.

8.4 Define a Mapping Class

In DynamoDB, a database is a collection of tables. 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 our bookstore app, each item in the table will represent a book, and each item will have five attributes: *Title*, *Author*, *Price*, *ISBN*, and *Hardcover*.

Each item (Book) in the table will have a hash key—in this case, ISBN—which is the primary key for the table.

We're going to map each item in the Book table to a `Book` object in the Java code, so that we can directly manipulate the database item through its object representation.

To establish mappings, DynamoDB defines annotations, including the following:

- **`@DynamoDBTable`**—Identifies the target table in DynamoDB.
- **`@DynamoDBHashKey`**—Maps a class property to the hash attribute of the table.
- **`@DynamoDBAttribute`**—Maps a class property to an item attribute.

For a complete list of the annotations that the Object Mapper offers, see Java Annotations for DynamoDB.

Let's create a `Book` mapping class:

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

@DynamoDBTable(tableName = "Books")
public class Book {
    private String title;
    private String author;
    private int price;
    private String isbn;
    private Boolean hardCover;

    @DynamoDBIndexRangeKey(attributeName = "Title")
    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }

    @DynamoDBIndexHashKey(attributeName = "Author")
    public String getAuthor() {
        return author;
    }

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

    @DynamoDBAttribute(attributeName = "Price")
    public int getPrice() {
        return price;
    }

    public void setPrice(int price) {
        this.price = price;
    }

    @DynamoDBHashKey(attributeName = "ISBN")
    public String getIsbn() {
        return isbn;
    }

    public void setIsbn(String isbn) {
        this.isbn = isbn;
    }

    @DynamoDBAttribute(attributeName = "Hardcover")
    public Boolean getHardCover() {
        return hardCover;
    }

    public void setHardCover(Boolean hardCover) {
        this.hardCover = hardCover;
```

```
    }
}
```

Note that `hardCover` is a nullable type. With the DynamoDB Object Mapper, primitives and nullable types behave differently. On a `save()`, an unset nullable type is not sent to DynamoDB; an unset primitive is sent as its default value.

8.5 Interact with Stored Objects

Now that we have a database, a mapping class, and an Object Mapper client, we can start interacting with objects in the cloud. These calls are synchronous and must be taken off of the main thread. You can wrap the code with:

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

8.5.1 Save an Item

To save an object, first create it and set the appropriate fields:

```
Book book = new Book();
book.setTitle("Great Expectations");
book.setAuthor("Charles Dickens");
book.setPrice(1299);
book.setIsbn("1234567890");
book.setHardCover(false);
```

Then use the Object Mapper client to write the object to a corresponding item in the table. In this case, we'll call `save()` on the client and pass in our `book` object:

```
mapper.save(book);
```

Except for the primary key (here "ISBN"), there is no predefined schema for the items in a table. We can update our mapping class and add or remove attributes at will. An item can have any number of attributes, although there is a limit of 400 KB on the item size.

8.5.2 Retrieve an Item

Using an object's primary key (in this case, the hash attribute "ISBN"), we can load the corresponding item from the database. The following code snippet returns the Book item with an ISBN of "1234567890":

```
Book selectedBook = mapper.load(Book.class, "1234567890");
```

8.5.3 Update an Item

To update an item in the database, just set new attributes and save the object again. For example, we could update the price of a Book instance as follows:

```
Book selectedBook = mapper.load(Book.class, "1234567890");
selectedBook.setPrice(1199);
mapper.save(selectedBook);
```

Note that setting a new hash key creates a new item in the database, even though it doesn't create a new object on the client side. Consider the following example:

```
Book selectedBook = mapper.load(Book.class, "1234567890");
selectedBook.setIsbn("0987654321");
mapper.save(selectedBook);
```

The result is a new item in the database, identical to the loaded item but with the new ISBN. The reference `selectedBook` now maps to this new item in the database, but the old item also exists.

8.5.4 Delete an Item

To delete an item from the database, use the `delete()` method and pass in the object to be deleted:

```
mapper.delete(selectedBook);
```

8.6 Perform a Scan

With a scan operation, we can retrieve all items from a given table. A scan examines every item in the table and returns the results in an undetermined order:

```
DynamoDBScanExpression scanExpression = new DynamoDBScanExpression();
PaginatedScanList<Book> result = mapper.scan(Book.class, scanExpression);
// Do something with result.
```

The returned list of items is lazily loaded when possible, so calls to DynamoDB are made only as needed. When you need to download an entire dataset in advance, you can call the `size()` method on the list to fetch the entire list.

The list returned by the Object Mapper can't be modified, and an attempt to do so results in an exception. If you want to use the result of a scan as a data source for a modifiable user interface component (for example, an editable `ListActivity`), you'll need to create a modifiable list object and move all of the data to it.

Scan is an expensive operation and should be used with care to avoid disrupting higher priority traffic on the table. The *Amazon DynamoDB Developer Guide* has Guidelines for Query and Scan that explain best practices for scan operations.

8.7 Perform a Query

A query operation lets us find items in a table using both hash and range key attributes. The primary key for our Books table doesn't have a range key. However, when we created the table, we specified a global secondary index, and that secondary index does have a range key attribute. We'll perform a query against the hash key and the range key of our secondary index.

8.7.1 Secondary Indexes

A secondary index is a data structure that contains a subset of attributes from a table, along with an alternate key to support query operations. With a secondary index, queries are no longer restricted to the table primary key; we can retrieve data using the alternate key, too.

The data in a secondary index consists of attributes that are projected, or copied, from the table into the index. Every secondary index is automatically maintained by DynamoDB. When we add, modify, or delete items in the table, any indexes on the table are also updated to reflect these changes.

To learn more about secondary indexes, see Improving Data Access with Secondary Indexes.

8.7.2 Query Example

The following example performs a query for books by the author "Charles Dickens" with a title beginning with "Great":

```
Book bookToFind = new Book();
bookToFind.setAuthor("Charles Dickens");

String queryString = "Great";

Condition rangeKeyCondition = new Condition()
        .withComparisonOperator(ComparisonOperator.BEGINS_WITH.toString())
        .withAttributeValueList(new AttributeValue().withS(queryString.
↪toString()));

DynamoDBQueryExpression queryExpression = new DynamoDBQueryExpression()
        .withHashKeyValues(bookToFind)
        .withRangeKeyCondition("Title", rangeKeyCondition)
        .withConsistentRead(false);

PaginatedQueryList<Book> result = mapper.query(Book.class, queryExpression);
// Do something with result.
```

We begin by creating a book object and setting the hash key attribute that we want to query against. The global secondary index for our Books table uses Author as a hash key, so we set the Author attribute for the Book item we're looking for.

Then we create a range key condition, which represents the selection criteria for our query. In this case, we want to select attribute values beginning with the string "Great".

When we create `DynamoDBQueryExpression`, we set the hash key value and the range key condition for the query. Note that the first parameter to `withRangeKeyCondition` is the range key attribute name.

Finally, we create a `PaginatedQueryList<T>` to represent the results from the query. Like the scan result list, the query result list can't be modified.

8.8 Conditional Writes

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 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 item has a "Price" value of "1299":

```
try {
    HashMap<String, AttributeValue> primaryKey = new HashMap<>();
    AttributeValue isbn = new AttributeValue()
            .withS("1234567890");
    primaryKey.put("ISBN", isbn);

    UpdateItemRequest request = new UpdateItemRequest()
            .withTableName("Books")
            .withKey(primaryKey)
            .addAttributeUpdatesEntry(
                    "Price", new AttributeValueUpdate()
                            .withValue(new AttributeValue().withN("1199"))
                            .withAction(AttributeAction.PUT))
            .addExpectedEntry(
                    "Price", new ExpectedAttributeValue()
                            .withValue(new AttributeValue().withN("1299"))
                            .withComparisonOperator(ComparisonOperator.EQ));

    ddbClient.updateItem(request);

}
catch (ConditionalCheckFailedException e) {
    // The conditional check failed.
}
```

In this example, we construct an UpdateItemRequest to pass to `updateItem()` on the DynamoDB client. The `UpdateItemRequest` object calls `addAttributeUpdatesEntry`, which specifies the name of the attribute to update, the new value for the attribute, and the action to perform on the attribute. To add a condition, we also call `addExpectedEntry`, which is the conditional block for the operation. In this case, the `ComparisonOperator` is checking that the price of the item equals (`EQ`) "1299". If this is not the case, the update fails.

Note that conditional writes are idempotent. This means that you can send the same conditional write

request multiple times, but it will have no further effect on the item after the first time DynamoDB performs the specified update.

8.9 Batch Operations

The DynamoDB Object Mapper provides batch write operations to put items in the database and delete items from the database. The following example illustrates a batch put operation using the `batchSave` method:

```
Book book1 = new Book();
book1.setTitle("Moby-Dick; or, The Whale");
book1.setAuthor("Herman Melville");
book1.setPrice(999);
book1.setIsbn("7654321098");
book1.setHardCover(false);

Book book2 = new Book();
book2.setTitle("Madame Bovary");
book2.setAuthor("Gustave Flaubert");
book2.setPrice(1099);
book2.setIsbn("6543210987");
book2.setHardCover(true);

Book book3 = new Book();
book3.setTitle("The Brothers Karamazov");
book3.setAuthor("Fyodor Dostoyevsky");
book3.setPrice(1399);
book3.setIsbn("5432109876");
book3.setHardCover(false);

mapper.batchSave(Arrays.asList(book1, book2, book3));
```

The `batchSave` method saves items into the database. We can use `batchDelete` to delete items from the database and `batchWrite` to either save or delete items.

CHAPTER 9

Process Streaming Data with Amazon Kinesis and Firehose

9.1 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.

The AWS Mobile SDK for Android provides simple, high-level clients designed to help you interface with Amazon 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.

9.2 What is Firehose?

Amazon Kinesis Firehose is a fully managed service for delivering real-time streaming data to destinations such as 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.

KinesisFirehoseRecorder is the high level client for Firehose. Its usage is very similar to that of *KinesisRecorder*.

For more information about Firehose, see Amazon Kinesis Firehose.

You can also learn more about how the Kinesis services work together on the following page: Amazon Kinesis services.

9.3 Getting Started

9.3.1 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 *Authenticate Users with Amazon Cognito Identity*.

9.3.2 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.

9.3.3 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.

9.3.4 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;
```

9.4 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.

```
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
```

9.4.1 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()`:

```
Long bytesUsed = recorder.getDiskBytesUsed();
// 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.

9.5 Use KinesisFirehoseRecorder

To use `KinesisFirehoseRecorder`, you need to pass the object in a directory where streaming data is saved. It's recommended to use an app private directory because the data isn't encrypted.

```
// 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();
```

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.

CHAPTER 10

Execute Code On Demand with Amazon Lambda

10.1 What is AWS Lambda?

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

The Mobile SDK for Android enables you to call Lambda functions from your Android mobile apps, using the `aws-android-sdk-lambda` library. When invoked from the Mobile SDK, Lambda functions receive data about the device, app, and end user identity through a client and identity context object, making it easy to create rich, and personalized app experiences.

The getting started section for AWS Lambda, To use AWS Lambda, *Mobile Backend Using Amazon Lambda*, provides step by step instructions for integrating the AWS Mobile SDK for Android into your app.

The AWS Mobile SDK for Android enables you to easily map client methods to Lambda function executions, using the @LambdaFunction annotation. For example, the code below will result in the "echo" Lambda function being executed every time the echo method is called inside the application code:

```
@LambdaFunction
NameInfo echo(NameInfo nameInfo);
```

The @LambdaFunction annotation 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. For example, the code below will result in the "echo" Lambda function being executed every time the noEcho method is called inside the application code:

```
@LambdaFunction(functionName = "echo")
NameInfo noEcho(NameInfo nameInfo);
```

invocationType Specifies how the Lambda function will be invoked. Can be one of the following values:

- Event – calls are one-way and asynchronous

- RequestResponse – calls take input and return output and are executed synchronously
- DryRun – allows you to validate access to a Lambda Function without executing it

LogType This parameter is valid only when invocationType is set to "RequestResponse". Valid values are "None" or "Tail". If set to "Tail" AWS Lambda will return the last 4KB of log data produced by your Lambda Function in the x-amz-log-results header, which is base64-encoded.

The class containing methods with the @LambdaFunction annotation must be instantiated using the `lambdainvoker.LambdaInvokerFactory.Build()` method. At the time of creating the `lambdainvoker.LambdaInvokerFactory` object, you will also be able to specify the region in which the Lambda function exists, and the Cognito Credentials provider to use to verify permissions to invoke the Lambda function.

10.1.1 Client and Identity Information

When invoked through the SDK, Lambda functions automatically have access to the data about the device and the app. It also has access to the end user identity by virtue of using Amazon Cognito as the credential provider. These are available in form of context.clientContext and context.identity. To learn more about AWS Lambda's programming model, see AWS Lambda.

You can access the client context within your lambda function as follows:

```
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_
↪versioin);
    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;
}
```

10.1.2 Client Context

client.installation_id Auto-generated UUID that is created the first time the app is launched. This is stored in shared preferences on the device. In case the shared preferences is wiped a new appId will be generated.

client.app_version_code versionCode from the Android Manifest

client.app_version_name versionName from the Android Manifest

client.app_package_name package name from the Android Manifest file

client.app_title Text defined as Android:label

env.platform_version Build.VERSION.RELEASE

env.platform Hardcoded as "Android"

env.make Build.MANUFACTURER

env.model Build.MODEL

env.locale Locale.getDefault()

10.1.3 Identity Context

To invoke a Lambda function from your mobile app, you can use Cognito as the credential provider. For more information, see Amazon Cognito. Cognito assigns each user a unique Identity ID. This Identity ID is available to you in the Lambda functions invoked through the AWS Mobile SDK. You can access the Identity ID as follows:

```
exports.handler = function(payload, context) {
    console.log("clientID = " + context.identity.cognitoIdentityId);
    context.succeed("Your client pool ID is " + context.identity.
↪cognitoIdentityIdPoolId);
}
```

10.1.4 Data Types

A method, annotated with LambdaFunction, can have at most one argument. When invoked, its argument is serialized into JSON. The invocation is translated to an AWS request and is sent to AWS Lambda service. After execution, Lambda returns a JSON encoded response which is deserialized into an object whose type matches the return type of the method. The (de)serialization is handled by LambdaDataBinder. The default implementation is LambdaJsonBinder backed by Gson.

```
public interface MyInterface {
   /*
    * String[] words = {"Hello", "world", "!"} is serialized as
    * ["Hello", "world", "!"]
    */

   @LambdaFunction
   String echo(String[] words);

   /*
    * NameInfo nameInfo = new NameInfo();
    * nameInfo.firstName = "John";
    * nameInfo.lastName = "Doe";
    * Then nameInfo is serialized as
    * {"firstName":"John","lastName":"Doe"}
```

```
     */
    @LambdaFunction
    String echo(NameInfo nameInfo);

    class NameInfo {
        String firstName;
        String lastName;
    }
}
```

In case you need to customize LambdaJsonBinder, you have the option to provide your implementation with LambdaInvokerFactory.build(Class<T>, LambdaDataBinder).

```
public class JacksonDataBinder implements LambdaDataBinder {
    private final ObjectMapper mapper;

    public JacksonDataBinder() {
        mapper = new ObjectMapper();
        mapper.setPropertyNamingStrategy(
            PropertyNamingStrategy.CAMEL_CASE_TO_LOWER_CASE_WITH_UNDERSCORES);
    }

    @Override
    public <T> T deserialize(byte[] content, Class<T> clazz) {
        try {
            return mapper.readValue(content, clazz);
        }
        catch (IOException e) {
            throw new AmazonClientException("Failed to deserialize content",␣
↪e);
        }
    }

    @Override
    public byte[] serialize(Object object) {
        try {
            return mapper.writeValueAsBytes(object);
        }
        catch (IOException e) {
            throw new AmazonClientException("Failed to serialize object", e);
        }
    }
}

// create a Lambda proxied object
MyInterface myInterface = lambdaInvokerFactory.build(
    MyInterface.class, new JacksonDataBinder());
```

10.1.5 Error Handling

When you invoke a method annotated with LambdaFunction and it results in an error on the server side, a LambdaFunctionException, subclass of RuntimeException, will be thrown. You can get the error message and the invocation result from the exception object.

Note that the method can fail due to other reasons, such as invalid credentials, network problem, or (de)serialization issue. These errors won't be turned into LambdaFunctionException.

```
// suppose echo(String) is an annotated Lambda function
try {
    String result = myInterface.echo("Hello world!");
}
catch (LambdaFunctionException lfe) {
    // Lambda code has error.
    Log.e(TAG, String.format(
        "echo method failed: error [%s], payload [%s].",
        lfe.getMessage(), lfe.getPayload());
}
catch (AmazonServiceException ase) {
    // invalid credentials, incorrect AWS signature, etc
}
catch (AmazonClientException ace) {
    // Network issue
}
```

For more information about Identity ID, see Cognito Identity.

CHAPTER 11

Understand Natural Language and Trigger Business Workflows with Amazon Lex

11.1 What is Amazon Lex?

Amazon Lex is an AWS service for building conversational interfaces into applications using voice and text. With Amazon Lex, the same deep learning 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. Amazon Lex provides the deep functionality and flexibility of natural language understanding (NLU) and automatic speech recognition (ASR) to enable you to build highly engaging user experiences with lifelike, conversational interactions and create new categories of products.

Amazon Lex enables any developer to build conversational chatbots quickly. With Amazon Lex, no deep learning expertise is necessary—you just specify the basic conversation flow in the Amazon Lex console to create a bot. Amazon Lex manages the dialogue and dynamically adjusts the responses in the conversation. Using the console, you can build, test, and publish your text or voice chatbot. You can then add the conversational interfaces to bots on mobile devices, web applications, and chat platforms (for example, Facebook Messenger).

Amazon Lex provides pre-built integration with AWS Lambda, and you can easily integrate with many other services on the AWS platform including Amazon Cognito, AWS Mobile Hub, Amazon CloudWatch, and Amazon DynamoDB. Integration with AWS Lambda provides bots access to pre-built serverless enterprise connectors, to link to data in SaaS applications, such as Salesforce, HubSpot or Marketo.

Using the following resources, you can integrate Amazon Lex with your Android app to add a flexible natural language interface for voice and text chat. No deep knowledge of either AWS services or natural language computing is needed.

For information on Amazon Lex concepts and service configuration, see How it Works in the *Lex Developer Guide*.

For instructions on how to integrate Amazon Lex into your Android application, see *Adding Natural Language Speech and Text to your App*.

For end to end sample apps using Amazon Lex:

- See Amazon Lex sample app in the AWS SDK for Android samples.
- Use the Conversational Bots feature in AWS MobileHub to quickly configure and provision a

Amazon Lex backend in the cloud. You can then download a working demonstration Android app project and SDK and helper files, all customized to match your configuration choices.

CHAPTER 12

Add Text to Speech Capability to your Android App with Amazon Polly

12.1 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 Android 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*.

For instructions on how to integrate Amazon Polly into your Android application, see Adding Text to Speech Conversion to your App.

For end to end sample apps using Amazon Polly see the AWS SDK for Android samples.

CHAPTER 13

Create Push Notification Campaigns with Amazon Pinpoint

13.1 What is Amazon Pinpoint?

Using Amazon Pinpoint, you can create push notification campaigns that provide your users with timely, relevant, personalized information to encourage them to keep using your mobile app. You can use push notification campaigns to increase app awareness, downloads, and launches; build customer loyalty; and ultimately boost your mobile revenues.

Campaigns can include things such as welcome and onboarding messages, invites to less-engaged or lapsed users, location-based invites, time-based reminders, and special offers for frequent users. You can use Amazon Pinpoint to create and push different campaigns based on groupings of specific user characteristics, which we refer to as user segments.

Using the following resources, you can integrate Amazon Pinpoint with your Android app to make it ready to be part of a campaign.

For information on Amazon Pinpoint concepts and service configuration, see the Amazon Pinpoint Developer Guide.

For instructions on how to integrate Amazon Pinpoint into your Android application, see Integrating Amazon Pinpoint With Android Apps.

For end to end sample apps using Amazon Pinpoint see the AWS SDK for Android samples.

CHAPTER 14

Additional Resources

- The AWS Mobile SDK for Android can be installed here.
- For more information about the AWS SDK for Android, including a complete list of supported AWS products, see the AWS Mobile SDK product page.
- The SDK reference documentation includes the ability to browse and search code included with the SDK. It provides thorough documentation and usage examples. You can find it at AWS SDK for Android API Reference.
- Post questions and feedback at the Mobile Developer Forum.
- Source code and sample applications are available at AWS Mobile SDK for Android sample repository.