# Build a WhatsApp clone for Android using AWS

With Amazon Web Services – or AWS – developers get a powerful platform that allows them to easily create a WhatsApp-like messaging application. Here, we will show how you can develop a social messaging app using a variety of AWS technologies.

**AWS Sample Messenger Features**

A typical messaging application should include features such as:

1. Log into the app using Facebook or Google+ accounts
2. Create chat rooms and invite friends from the address book
3. Send/Receive messages
4. Store and share pictures
5. Measure adoption with key metrics

Most functionalities in a messaging application require a lot of work, both inside the app and at the backend. For starters, the developer has to write the code to integrate sign-in functionality of each of the social media platforms; he/she also has to create an infrastructure that is capable of handling millions of chat rooms and billions of messages every day. In addition to all this, he/she must also keep track of consumers' usage patterns to enhance the ease-of-use and maximize revenue.

With the strength of its infrastructure, AWS makes it extremely easy for developers to handle all these processes. By providing a single hub to work with, and automating a lot of the infrastructural work, AWS significantly cuts down on app development time; it reduces time needed to market the app, and minimizes operational cost.

To give you an idea of how AWS simplifies app development, we will list the various AWS technologies that we'll be using to build our messaging app.

***AWS Mobile Hub***

To quickly setup, configure and customize features that will make our Android application dynamic. AWS Mobile Hub will also help us cut down on development time.

***Amazon Cognito Identity***

To allow users to log into our messaging app, using their Facebook and/or Google+ accounts

***Amazon Mobile Analytics***

To capture user actions for UX analysis

***Amazon DynamoDB***

To create and manage chat rooms, and store user messages

***Amazon S3***

To store the images shared by users in chat rooms

***Amazon SNS***

To send Push Notifications to chat room participants when a new message is sent to the chat room

***Amazon Device Farm***
To test sample application across multiple devices

**Prerequisites**
- Prior knowledge of app development using Android OS
- AWS account

**System Requirements**
- Android Studio 1.4 or newer
- Android SDK 4.4 (KitKat) API Level 19 or newer
- Android SDK Build-tools 23.0.1
- Android Device with Android OS 4.0.3 (IceCream Sandwich) API Level 15 or newer

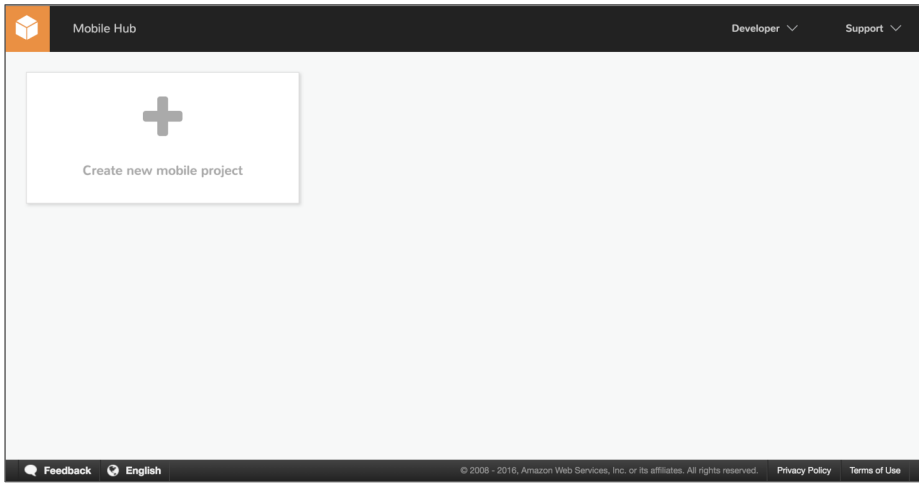
## 1. Project Setup

With this background, we are now ready to start developing our application. We will begin with creating an Android project using **Amazon Mobile Hub**, which is a powerful console to configure complex projects easily and conveniently through a single interface.
*What is Amazon Mobile Hub?*

- Go to Mobile Hub Console and log onto your AWS developer account. The AWS Mobile Hub Console lets you to conveniently create, edit and delete your projects.

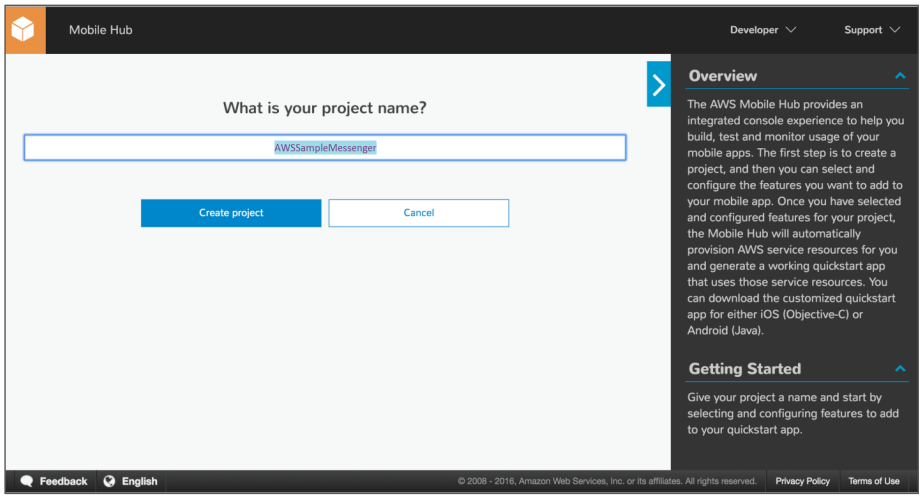
*I don't have an AWS developer account.* Create a new account.

- Select "Create new mobile project" from the dashboard




*Screen 1: AWS Mobile Hub Dashboard*

- Enter the name of your project. For this tutorial, we will be naming our project `AWSSampleMessenger`.




*Screen 2: Creating a new project*

- Now simply click on “Create Project” to create a basic project, ready for further configuration and addition of AWS functionality.

## 2. Social Networks Sign-in

.
Our sample application will allow users to sign-in using their existing social network accounts, such as Facebook and Google+. This can be easily done using **Amazon Cognito Identity** service. With the basic project in place, we can now add the *Cognito Identity* feature using Mobile Hub.

For more information about Cognito identity, visit Amazon Cognito

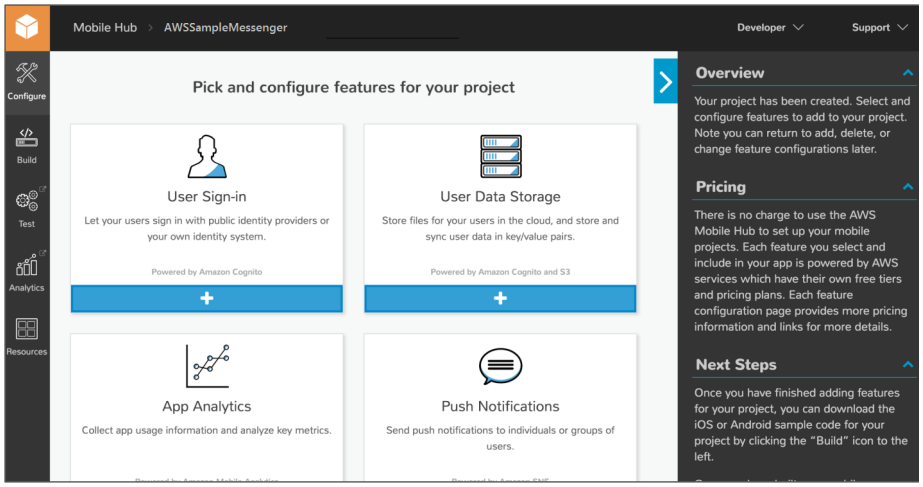
**What does it cost?**
For your first 50,000 monthly active users (MAUs), Amazon Cognito Identity is completely **free**.

Click here for information about pricing for MAUs above the free tier

**註解 [A1]:** Section added

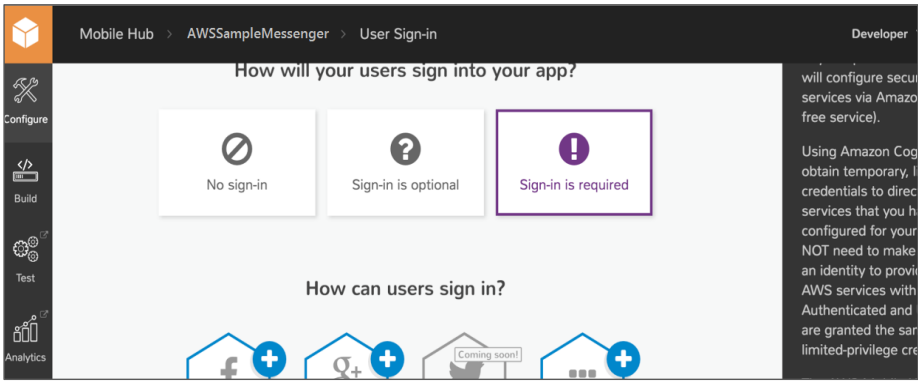
.

- From the Mobile Hub dashboard, select the project name and click on the + button on the "User Sign-in" box.



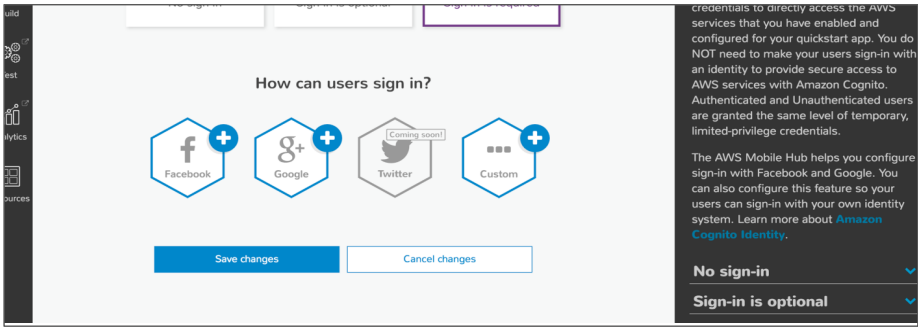

*Screen 3: Project customizable features*

- Since our application will enforce sign-in *before* anyone can use the application, we will select "Sign-in is required".



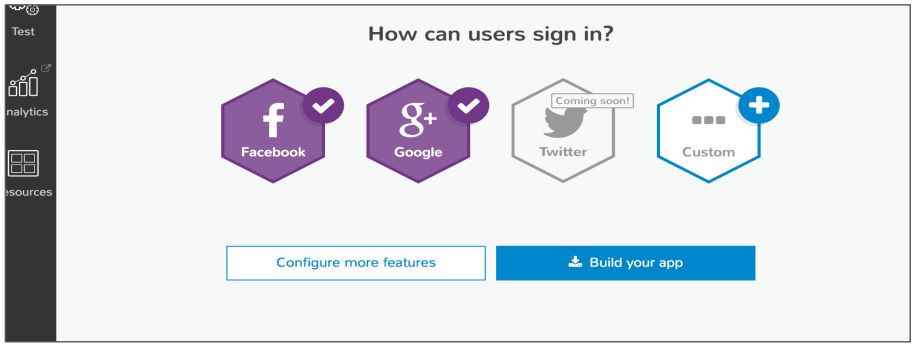

*Screen 4: Enabling social media log-in*

- To configure Facebook for your app, click on the + button on the Facebook logo and follow the instructions provided at: https://developers.facebook.com/docs/android/getting-started




*Screen 5: Configuring different social media networks*

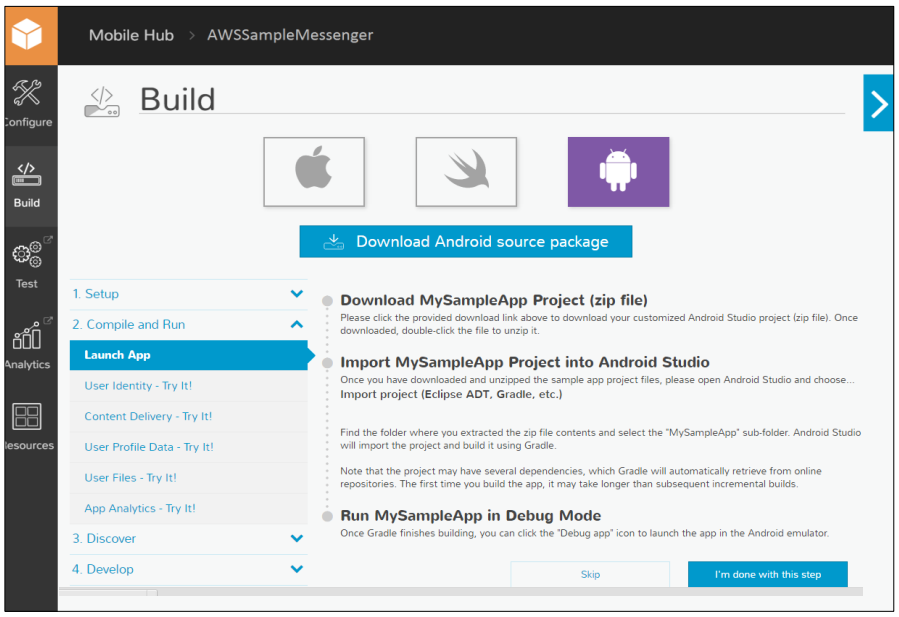
- To configure Google+ for your app, click on the + button on the Google+ logo and follow the instructions provided at: https://developers.google.com/+/mobile/android/getting-started
- After successfully configuring social media log-in, click on “Save Changes”.
- Once everything is configured successfully, click on “Build your app”.




*Screen 6: Building the app*

Note: At this stage, Mobile Hub has brought in the required *Cognito Identity* feature, but we still need to select the platform and the programming language. Mobile Hub supports Android, Objective-C based iOS and Swift-based iOS platforms. For this sample application, we want to use Android for programming.

- First select the box for the Android platform logo, then click on “Download Android source package”.




*Screen 7: Downloading the source package*

## How to use sample application source code

As you know, any application built with AWS services, needs to be associated with your Amazon Developer Account. Similarly, services including Facebook login, Google+ login, AWS SNS etc. require some configuration that is specific to you. The accompanying source code for the sample messenger application does not contain any reference to the Amazon Developer Account or other services. So before you can use this source code, you have to do some configuration. There are two ways of going about it:

### Option #1

- Downloaded the source package
- Within the source package, locate the file `AWSConfiguration.java`
- Open the file and fill in the empty strings for the specified keys:

```
// AWS MobileHub user agent string
    public static final String AWS_MOBILEHUB_USER_AGENT =
        "";
    // AMAZON COGNITO
    public static final Regions AMAZON_COGNITO_REGION =
        Regions.US_EAST_1;
    public static final String  AMAZON_COGNITO_IDENTITY_POOL_ID =
        "";
    // AMAZON MOBILE ANALYTICS
    public static final String  AMAZON_MOBILE_ANALYTICS_APP_ID =
        "";
    // Amazon Mobile Analytics region
    public static final Regions AMAZON_MOBILE_ANALYTICS_REGION =
Regions.US_EAST_1;
    // Google Client ID for Web application
    public static final String GOOGLE_CLIENT_ID =
            "";
    // GOOGLE CLOUD MESSAGING API KEY
    public static final String GOOGLE_CLOUD_MESSAGING_API_KEY =
            "";
    // GOOGLE CLOUD MESSAGING SENDER ID
    public static final String GOOGLE_CLOUD_MESSAGING_SENDER_ID =
            "";

    // SNS PLATFORM APPLICATION ARN
    public static final String AMAZON_SNS_PLATFORM_APPLICATION_ARN =
        "";
    // SNS DEFAULT TOPIC ARN
    public static final String AMAZON_SNS_DEFAULT_TOPIC_ARN =
        "";
    // SNS PLATFORM TOPIC ARNS
    public static final String[] AMAZON_SNS_TOPIC_ARNS =
        {""};
    public static final String AMAZON_CONTENT_DELIVERY_S3_BUCKET =
        "";
    // S3 BUCKET
    public static final String AMAZON_S3_USER_FILES_BUCKET =
         "";
```

**註解 [A2]:** Need to clarify if acc. Information can be added to empty strings all at once

- Locate the file `AndroidManifest.xml` and add your “facebook_app_id” in the following string:

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

- Create the following DynamoDB tables in Amazon Mobile Hub
  - UserProfile
  - ChatRoom
  - Conversation

Note: Attributes for these tables can be found in Section 4 of this tutorial, under ‘Sample Database Schema’

**Option#2**

- Create and configure your own project on Amazon Mobile Hub as described in this tutorial
- Download the source code for the AWS Sample Messenger Project.
- Drag and drop the Chat folder from the sample project into your own project
- Now open the file `SplashActivity.java` and perform activity calls to the following functions:

```
/**
     * Go to the ChatRoomHome activity after the splash timeout has expired.
     */
    protected void goMain() {
        Log.d(LOG_TAG, "Launching ChatRoomHome Activity...");
        goAfterSplashTimeout(new Intent(this, DashBoardActivity.class));
    }

    /**
     * Go to the log in activity after the splash timeout has expired.
     */
    protected void goLogin() {
        Log.d(LOG_TAG, "Launching Login-in Activity...");
        goAfterSplashTimeout(new Intent(this, LoginActivity.class));
    }
```

- Add Chat room related activities in `AndroidManifest.xml`. Also add Permissions for

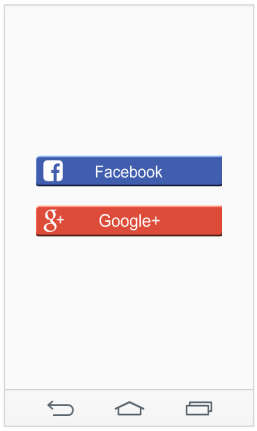
```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.READ_CONTACTS" />
```

- Add all Chat room related strings in `strings.xml`

- Once these steps are done, simply run the code.

At this stage, if you launch the application, you will automatically get a splash screen that Amazon Mobile Hub has generated for you. You will also be able to see the following sign-in screen [Screen 8] that allows you to login, using your Facebook and/or Google+ account(s) [depending on which network(s) you enabled during configuration].

Amazon Cognito Identity does the heavy lifting for you while still giving you the flexibility to customize these screens to meet your specific needs.




*Screen 8: Sample sign-in screen*

## 3. Amazon Mobile Analytics

So we have the sample project in place, and have created it to allow users to sign in from their social media accounts. At this stage, we need to incorporate another feature into our project; one that is a must-have for any consumer-facing application – Analytics. Among other things, an analytics feature would allow us to evaluate a user's app usage from different social media networks.

In-app analytic tools help generate useful data regarding an app's consumption, impression and reach. By incorporating analytics early into your app, you can effectively extract, quantify, organize and analyze data about important use cases, which can help you make data-driven decisions to monetize your app and increase user engagement.

**Amazon Mobile Analytics** helps you measure your app usage by automatically generating and compiling analytic reports. This data can help you track key trends about your users, evaluate and extrapolate app revenue, gauge user retention, and monitor custom in-app behavior events.

For more information about analytics, visit Amazon Mobile Analytics

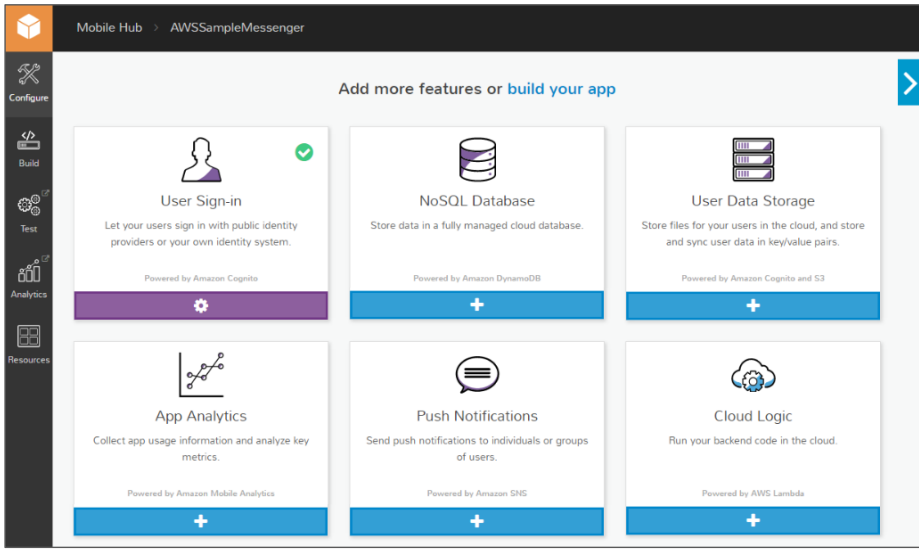
**What does it cost?**
For up to 100 million events per month, Mobile Analytics is completely **free**.

Click here for information about pricing for events above the free tier

註解 [A3]: Section added

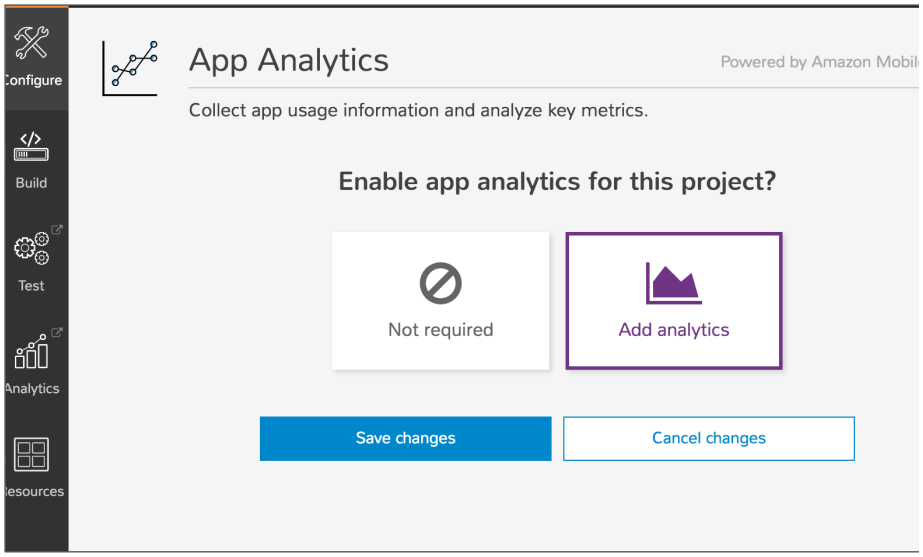
So let's begin by incorporating Amazon Mobile Analytics into our application. The goal is to track whether the user logged in using Facebook or Google+ account.

- In order to do that, go to the project dashboard on Mobile Hub and click on the + button on the "App Analytics" box.



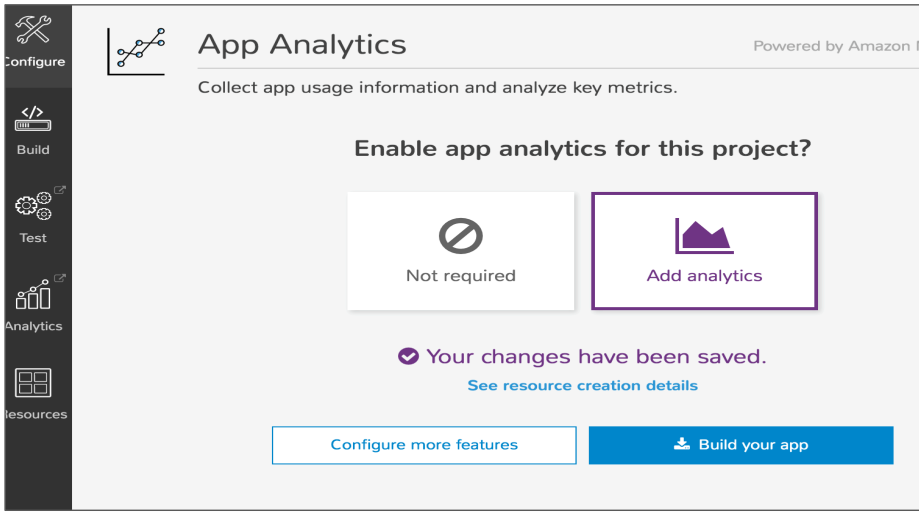

*Screen 9 Project features*

- On the resulting screen, select “Add Analytics”, and click on “Save Changes”.




*Screen 10 App Analytics feature*

- Once the changes are successfully saved, simply click on “Build your app” button.

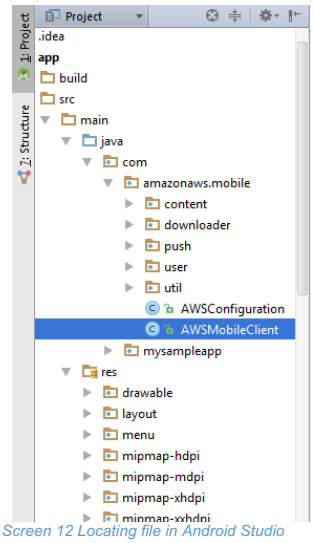



*Screen 11 Building the app after enabling App Analytics*

- Now, like we did earlier for Cognito Identity [Screen 7], choose the box with the Android platform logo and download source package.

So far we have added Amazon Mobile Analytics package in the Android project. Now we will add an Analytics-specific source code to the project.

- First open the downloaded project in Android Studio.
- Find and select filename “`AWSMobileClient.java`” from the project navigator.
- Now add the auto-generated code. This code assists in the tracking of user session data in Amazon Mobile Analytics.




*Screen 12 Locating file in Android Studio*

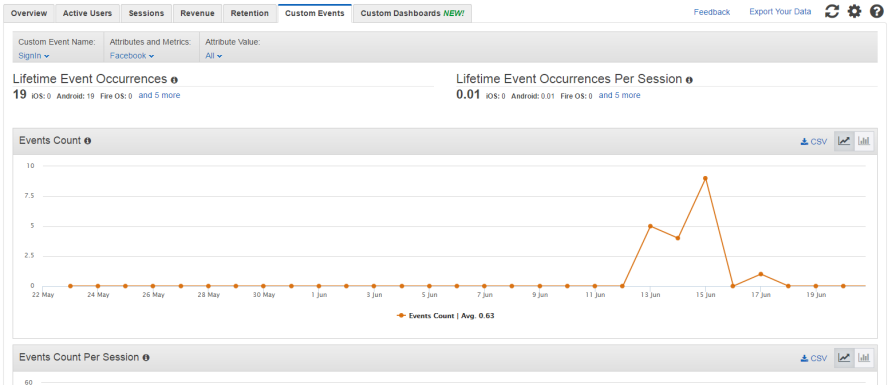
```
public void handleOnPause() {

        SessionClient sessionClient = null;
        EventClient eventClient = null;

        try {
            if (mobileAnalyticsManager != null &&
                (sessionClient = mobileAnalyticsManager.getSessionClient()) != null &&
                (eventClient = mobileAnalyticsManager.getEventClient()) != null) {
                sessionClient.pauseSession();
                eventClient.submitEvents();
            }
        }
        catch (final Exception e) {
            Log.w(LOG_TAG, "Unable to report analytics. " + e.getMessage(), e);
        }
    }
```

This is all you need to track the Facebook and Google+ login usage. You can now view the resulting data by going to the Analytics tab in the left panel in Mobile Hub [as shown in Screen 13] and view all the analytics, as shown in Screen 14.


*Screen 13: Analytics tab on Mobile Hub*

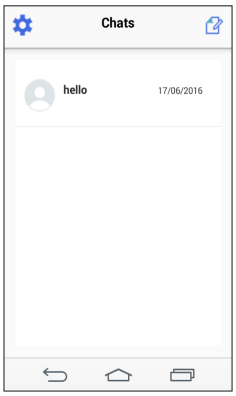



*Screen 14: App Analytics console*

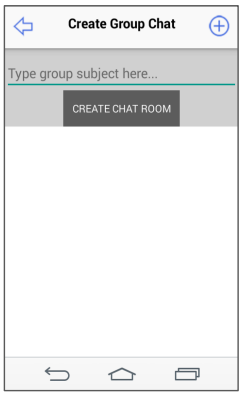
## 4. Amazon NoSQL Database

This brings us to the development of the main functionality for our messaging application – the ability to allow users to create chat rooms, add participants to individual chat rooms, and send/receive messages in a chat room. Our goal is to create a messaging app that, when logged into, welcomes the user with a dashboard screen similar to Screen 15.

*How does it work?*
The user can create a new chat room by pressing the icon on the top-right corner of the screen [as shown in Screen 16]. Similarly, when the user presses an individual chat room, we will show the detailed view for said chat room.




*Screen 15 Main dashboard*




*Screen 16:Individual chat room*

This application needs to be able to handle the potential of hundreds of thousands of active chat rooms, if not more. It also needs to be capable of handling millions of messages being sent and received simultaneously.

Traditional relational databases do not scale well in such situations. Hence, the solution is to use a NoSQL database; this is where **Amazon DynamoDB** comes in. Amazon DynamoDB is Amazon's cloud-based NoSQL Database

For more information about Amazon DynamoDB, visit Amazon DynamoDB.

**What does it cost?**
DynamoDB customers can get started with a **free tier**.

Visit DynamoDB Pricing for more information about all the pricing tiers available.

**註解 [A4]:** Section added

service, which allows you to create a database and have it managed by a service that stores and retrieves your in-app data. For our sample messenger, the *Amazon NoSQL Database* feature would allow us to store information about chat rooms, conversations, and chat room recipients.

## Sample Database Schema

To implement Amazon DynamoDB, we must first create a database schema. This database schema would help us define how the data would be organized, as well as the relationship between different aspects of the data. For this particular application, we need to create three different tables, each with a different set of attributes.

1. **UserProfile** (Protected)

| | | | |
|---|---|---|---|
| **userId(PK) -** String | **phone(SK)** – String | **Name** - String | **pushTargetArn –** String set |

2. **ChatRoom** (Protected)

| | | | | |
|---|---|---|---|---|
| **userId(PK) –** String | **chatRoomId(SK) –** String | **createdAt(SK) –** String | **name -** String | **recipients -** String set |

*Index name: ByCreationDate*
*Partition key: chatRoom Id – String*
*Sort key: createdAt – String*

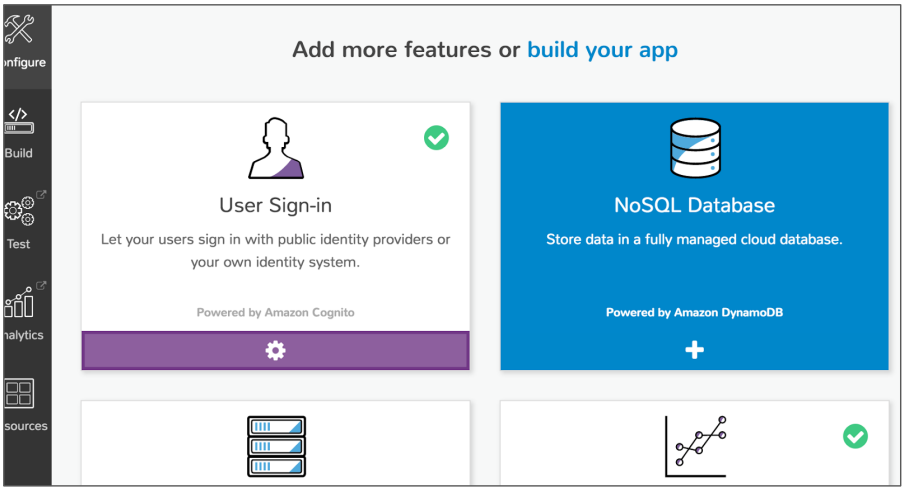
3. **Conversation** (Protected)

| | | | | |
|---|---|---|---|---|
| **userId(PK) –** String | **conversationId(SK) -** String | **createdAt –** String | **chatRoomId** - String | **message –** String |

*Index name: ByCreationDate*
*Partition key: chatRoom Id – String*
*Sort key: createdAt – String*

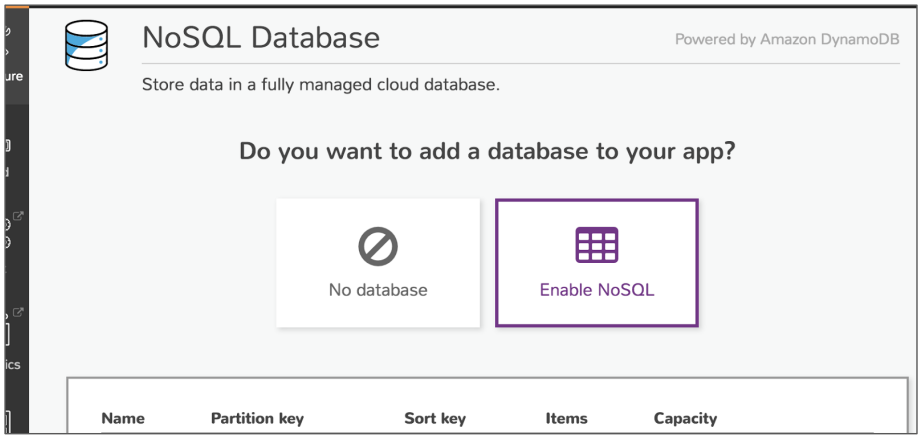
Now that we have prepared a database schema, we can integrate the *Amazon NoSQL Database* feature into our project.

- Open the project dashboard in Mobile Hub and go to its list of customizable features
- Click on the + sign on the “NoSQL Database” box



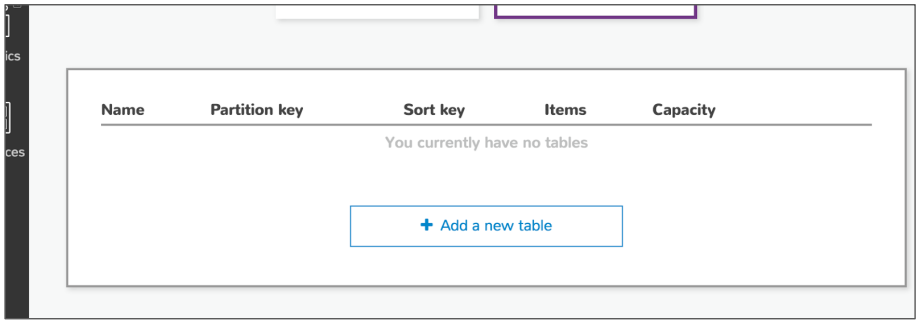

*Screen 17: NoSQL Database feature*

- Select the “Enable NoSQL” option on the resulting screen.




*Screen 18: Enabling NoSQL*

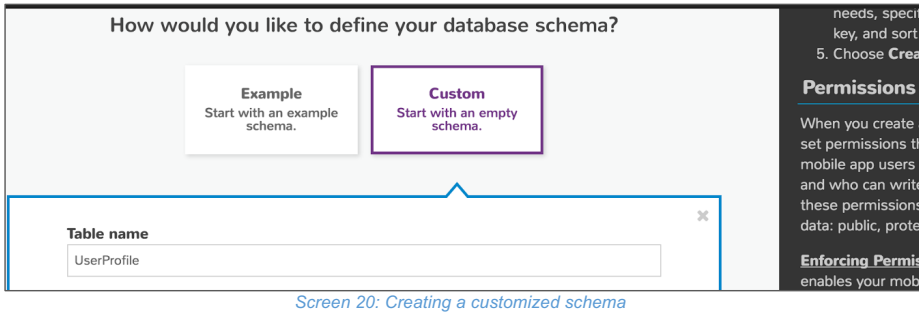
- Click on the “Add a new table” box in the section below




*Screen 19: Adding a table to the database*

- Now you must define your database schema. To proceed with a blank, customizable schema, select “Custom”.

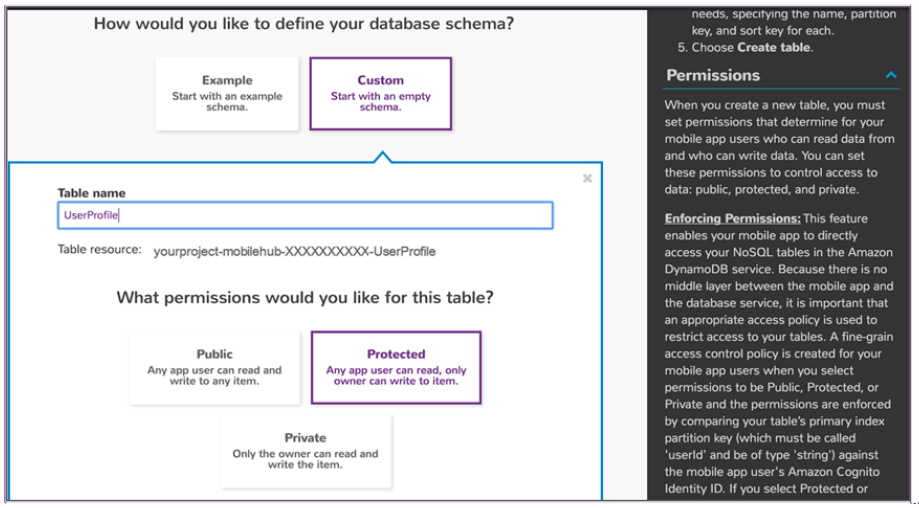
*How does this help?*
The database schema is the design you’ll be using to develop your database in NoSQL. For our messaging application, the database we develop will allow us to store chat room information and messaging history.




*Screen 20: Creating a customized schema*

- Enter the name of the first table you wish to create. As defined in Section 4, Sample Database Schema sub-section 1, we will write `UserProfile` in the field for Table name

- Now select the permission settings you’d like your database schema to have. Each permission option briefly describes the setting; details for the permission setting appear on the right sidebar. Select “Protected”




*Screen 21: Choosing permission setting for table*

註解 [A5]: Image modified to conceal ID

- Enter the attributes of the table. As shown in the Sample Database Schema section, add attributes for table name: `UserProfile`

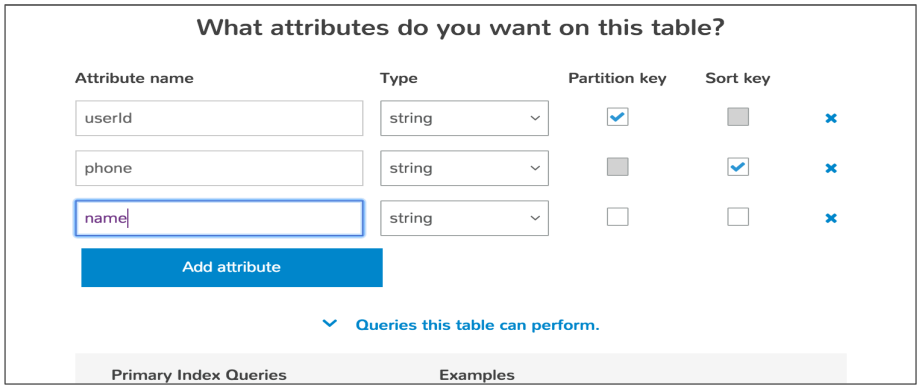
| **userId(PK) -** String | **phone(SK)** – String | **Name** - String | **pushTargetArn –** String set |
|---|---|---|---|

*First attribute*
Enter attribute name: `userId`. Set type as: `string`. Check box for Partition key

*Second attribute*
Enter attribute name: `phone`. Set type as: `string`. Check box for Sort key

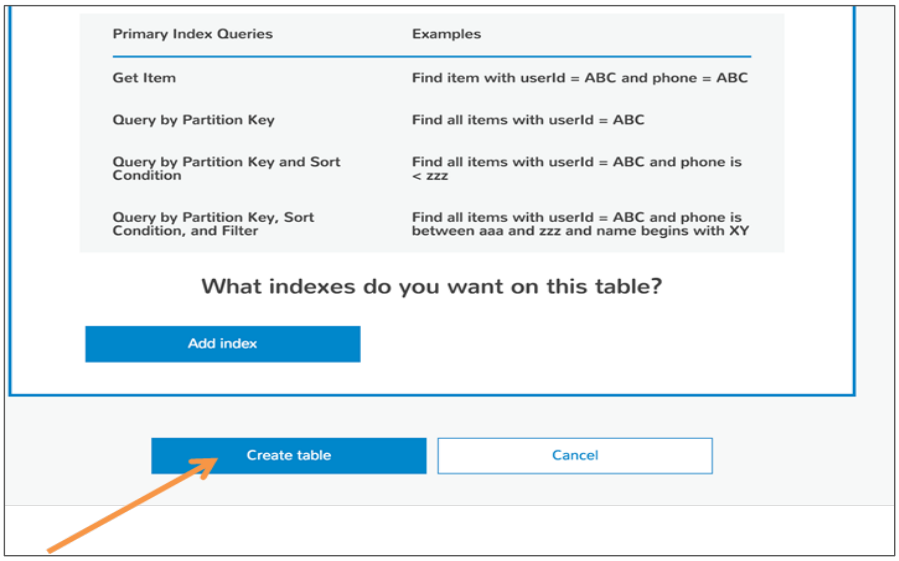
*Third attribute*
Enter attribute name: `name`. Set type as: `string`

*Fourth attribute*
Enter attribute name: `pushTargetArn`. Set type as: `string set`




*Screen 22: Adding table attributes*

- After entering all attributes for the table `UserProfile`, click on the “Create table” button appearing at the bottom of the screen

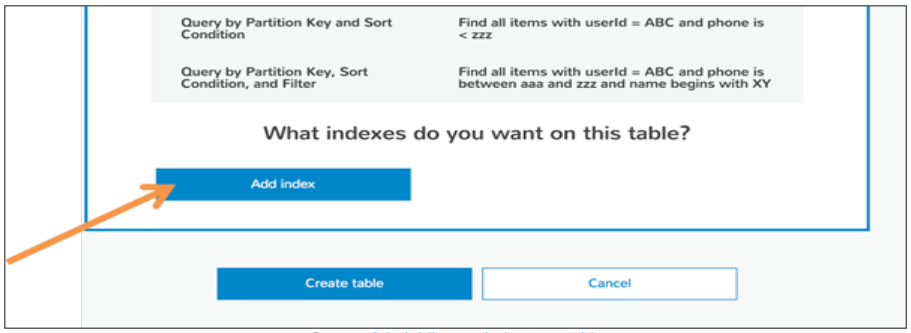



*Screen 23: Creating a table*

- Now repeat the same steps for tables `ChatRoom` and `Conversation`.

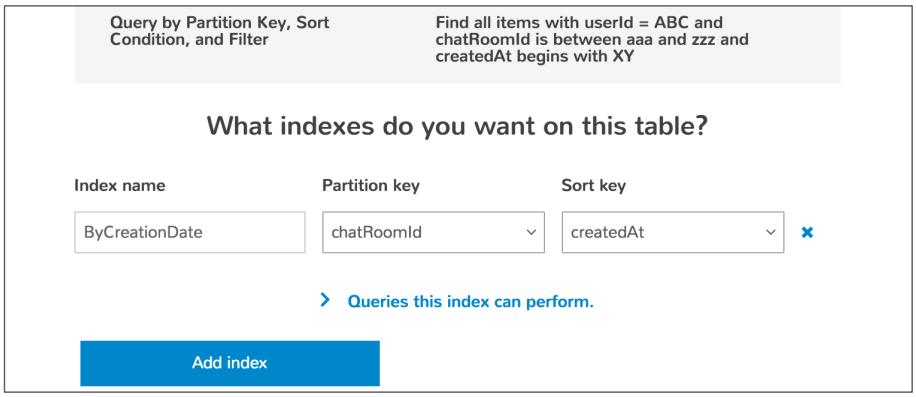
*Adding an index*

- To add index for a particular table, click on “Add index” after entering the table’s attributes but before clicking on “Create table”



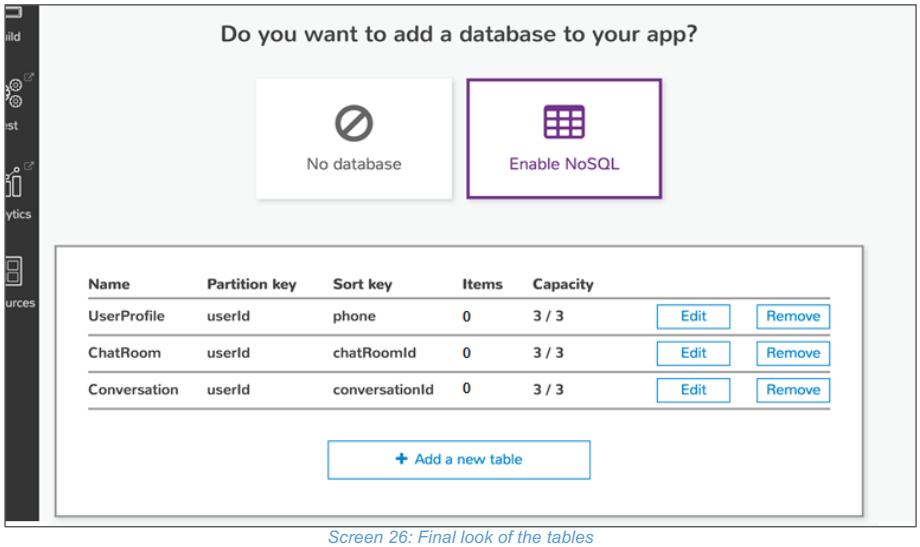

*Screen 24: Adding an index to a table*

- Select index attributes and click on “Add index” once again




*Screen 25: Selecting index attributes*

- Once all the steps are completed, this is how the tables should appear.




*Screen 26: Final look of the tables*

- Finally, build your app, download the Android source package and run the code.

### Creating Chat Rooms with Multiple Recipients

Once we have defined the database schema by creating the tables with their corresponding attributes, and have enabled NoSQL Database, we must enable our application to use that database to create chat rooms. Our messaging app needs to be able to allow users to easily create chat rooms and initiate conversations with one or multiple contacts from the device's address book. In order to successfully enable this, we must perform the following tasks:

- Create the function `loadUsersWithPhoneList` to get all selected recipients from the `UserProfile` table. Executing this code will allow us to load all the contacts from the logged-in user's database.

**Note**: In the accompanying sample application, while creating a new chat room, a user can only add recipients from his/her address book, who are already registered with the application, and who provided the same telephone number during their first sign-in, that exists in the user's address book.

```
public PaginatedScanList<UserProfileDO> loadUsersWithPhoneList(IdentityManager identityManager,
Object[] phNoList) {

        ddbClient = new AmazonDynamoDBClient(identityManager.getCredentialsProvider());
        mapper = new DynamoDBMapper(ddbClient);

        Map<String,  AttributeValue> filter = new HashMap<String, AttributeValue>();
```

```
        //search users on the base of phone number
        int i = 0;
        for ( Object obj : phNoList ) {
            filter.put(":val"+i++,new AttributeValue().withS(obj.toString()));
        }
        Set<String> filterkeySet = filter.keySet();
        String newSetStr = null ;
        for(String str : filterkeySet){
            if(newSetStr == null){
                newSetStr = str;
            }else{
                newSetStr += "," + str;
            }
        }

        final DynamoDBScanExpression scanExpression = new DynamoDBScanExpression()
                .withFilterExpression("phone in ("+newSetStr+")")
                .withExpressionAttributeValues(filter);

        PaginatedScanList<UserProfileDO> scanResult  = mapper.scan(UserProfileDO.class,
scanExpression);
        if(scanResult != null){
            return scanResult;
        }

        return null;

    }
```

- Create the function `saveNewChatRoom` to insert new chat room in `ChatRoom` table. This code would enable the app to allow the user to create a new chat room with one or more recipients from the previously loaded data.

```
//add chat room for group chat
    public void saveNewChatRoom(String dt, IdentityManager identityManager, String groupName,
Set<String> recipientId, String chatRoomId) {

        AmazonDynamoDBClient ddbClient = new
AmazonDynamoDBClient(identityManager.getCredentialsProvider());
        mapper = new DynamoDBMapper(ddbClient);

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

- Create function in `CreateChatRoomActivity` to call the two functions above in sequence.

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

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

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

                @Override
                protected void onPostExecute(String msg) {

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

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

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

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

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

                            @Override
                            protected void onPostExecute(String msg) {
                                Intent intent = new Intent(CreateChatRoomActivity.this,
ChatActivity.class);

                                intent.putExtra("ID",randomUUIDString);
                                intent.putExtra("Flag",false);
                                startActivity(intent);
                                finish();

                            }
                        }.execute();
```

```
                    }
                }
            }.execute();

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

### Load Chat Rooms on Screen

In the previous step, we allowed our app to create chat rooms, using information from the user's database. Now we must enable it to load all created chat rooms on one screen [as shown in Screen 15]. To allow our app to load a chat room onto the screen, we must create two functions

- We must first implement function `loadUserChatRooms` to load all associated chat rooms for a logged-in user

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

        ddbClient = new AmazonDynamoDBClient(provider.getCredentialsProvider());
        mapper = new DynamoDBMapper(ddbClient);

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

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



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

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

        return null;
    }
```

- Then, we implement function `showChatRooms` in `DashboardActivity` to show all chat rooms on screen

```
private void showChatRooms(){

        if(awsChatRoomsList.size() != 0){
            //change and set date format in ChatRoomDO
            for ( ChatRoomDO chatRoom : awsChatRoomsList) {
                String changeDate =
chatRoomListAdapter.setDateWithNewFormat(chatRoom.getCreatedAt());
```

```
                chatRoom.setCreatedAt(changeDate);
            }

            for ( ChatRoomDO chatRoom : awsChatRoomsList) {
                chatRoomListAdapter.add(chatRoom);
            }

            chatRoomListAdapter.sort();
            chatListView.setAdapter(chatRoomListAdapter);
            chatRoomListAdapter.notifyDataSetChanged();

            chatListView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
                @Override
                public void onItemClick(AdapterView<?> parent, View view, final int position, long
id){
                    //call conversation activity
                    Intent intent = new Intent(DashBoardActivity.this,ChatActivity.class);
                    intent.putExtra("ID",  chatRoomListAdapter.getId(position));
                    intent.putExtra("Flag",false);
                    startActivity(intent);
                }
            });
        }
```

**Download source code**

## 5. Enabling Amazon S3 feature from Mobile hub dashboard

In addition to creating a new chat room with one or more recipients, and viewing all active chat rooms on a single screen, we want our application to allow participants to send and receive images to and from each other. To be able to send and receive files such as images, our app needs to have a defined storage space. This is where Amazon's Simple Storage Service, or **Amazon S3**, comes in. For Amazon S3 to be successfully incorporated into our application, we must first enable our application's User Data Storage feature from Mobile Hub.

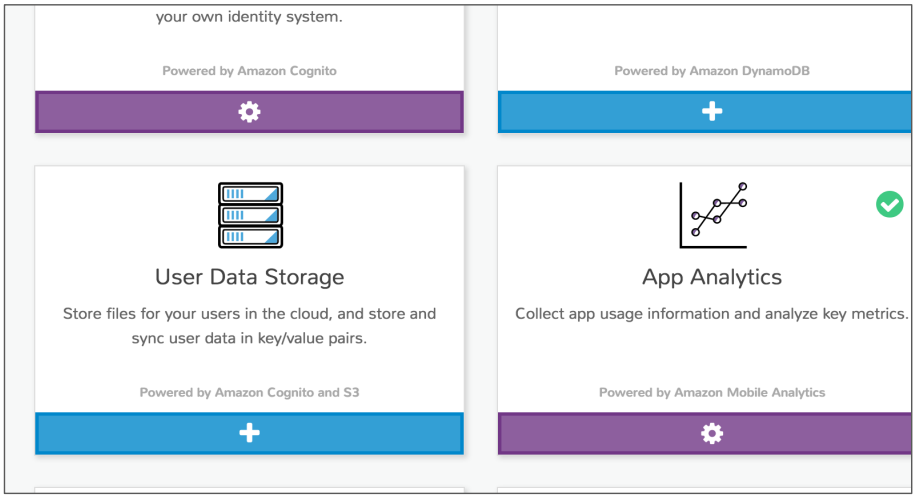
For more information about Amazon's Storage Service, visit Amazon S3.

**What does it cost?**
Amazon S3 customers can get started with a **free tier**.

Visit S3 Pricing for more information about all the pricing tiers available.

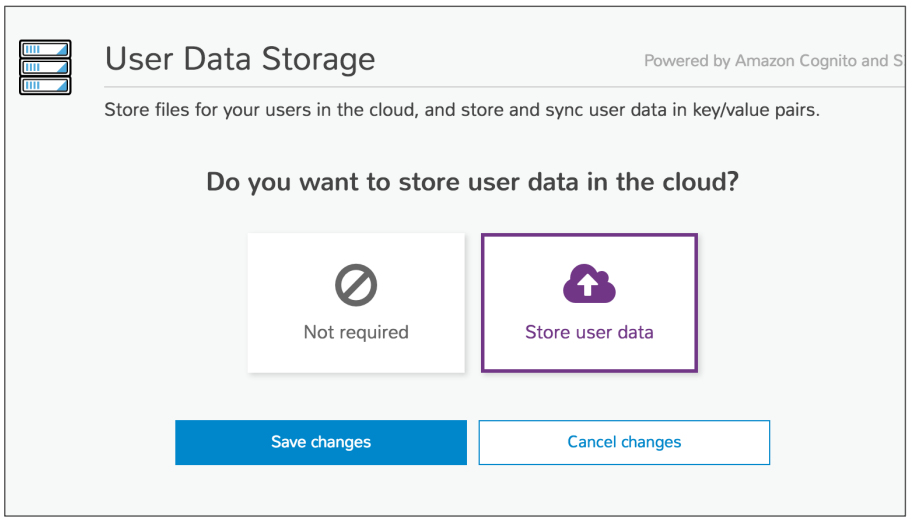
**註解 [A6]:** Section added

- Go to the project's features from Mobile Hub and click on the + sign on the "User Data Storage" box from feature list




*Screen 27: Selecting 'User Data Storage' feature*

- Select "Store user data" from the two options available, and click on "Save changes".

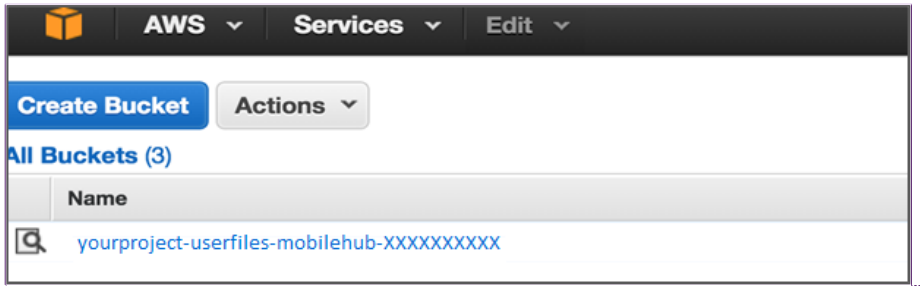



*Screen 28: Configuring the data storage feature*

**Adding policy to bucket public folder**

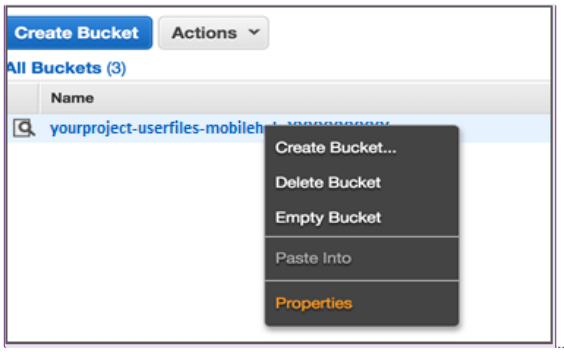
Open the Amazon S3 URL: https://console.aws.amazon.com/s3

- You will see a bucket name similar to `yourproject-userfiles-mobilehub-XXXXXXXXXX`. This bucket was created when you performed the previous step.




*Screen 29: Project bucket on S3 console*

- Right-click on the bucket name and go to “Properties”

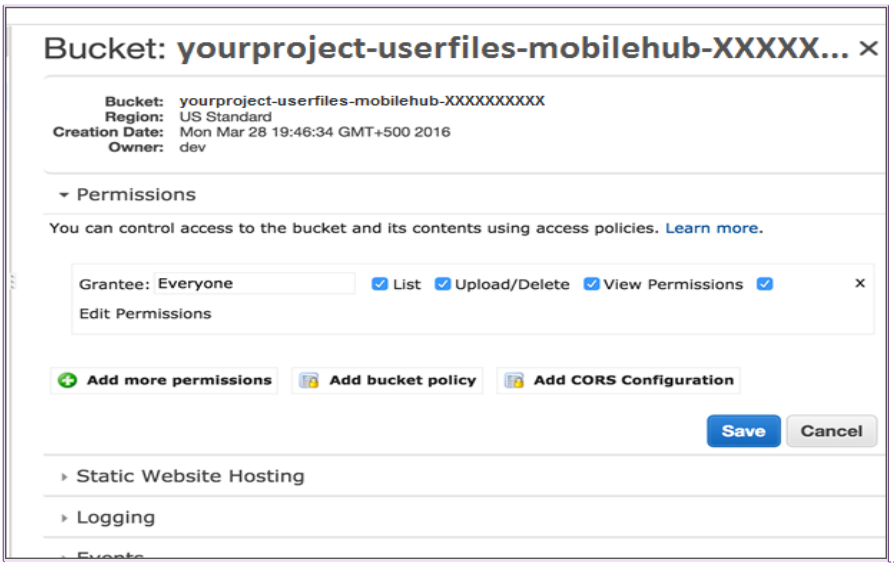



- The Properties will now open in a panel on the right side. Select “Permissions” and select “Add bucket policy”.

註解 [A7]: Was previously “awssamplemessenger-contentdelivery-mobilehub-xxxxxxx”

註解 [A8]: Image modified to conceal ID

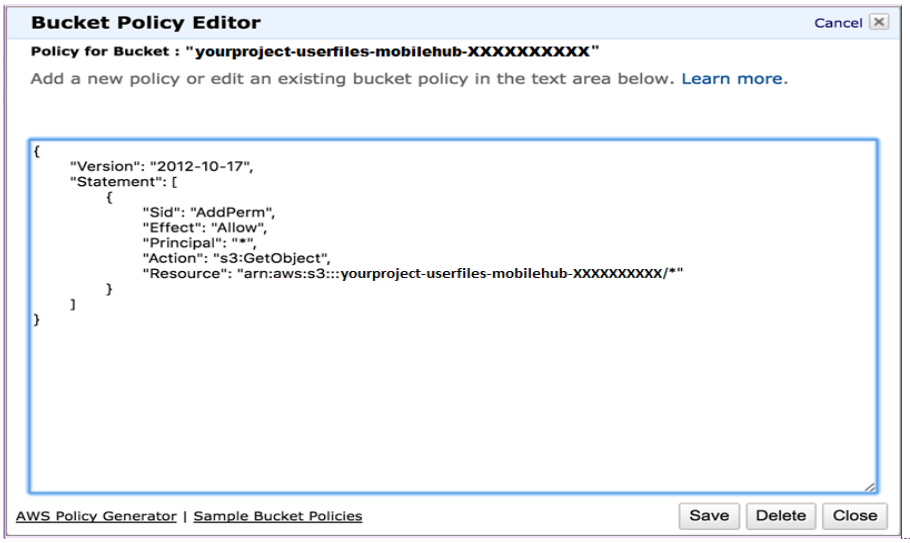
註解 [A9]: Image modified to conceal ID




註解 [A10]: Image modified to conceal ID

*Screen 30: Bucket "Permissions" panel*

- Enter the following permissions in the Policy Editor and click "Save".




註解 [A11]: Image modified to conceal ID

*Screen 31: Bucket Policy Editor*

### Uploading an image from code

If you want the user to be able to send a picture to another participant in the chat room, perform the following tasks:

- Get the image URI from `ImageSelectorUtils`
- Provide the file path to `UserFileManager.java`. The `UserFileManager` would upload the image by calling the method `uploadContent`.

Following is the code illustrating it:

```
private void uploadWithData(Intent data) {

        final Uri uri = data.getData();
        //get file path from a Uri
        final String path = ImageSelectorUtils.getFilePathFromUri(ChatActivity.this, uri);
        final File file = new File(path);
        //get an instance of User File Manager which uploads files from Amazon S3
        userFileManager.uploadContent(file,file.getName(), new ContentProgressListener() {
            @Override
            public void onSuccess(final ContentItem contentItem) {

                final AmazonS3 s3 =
                        new
AmazonS3Client(AWSMobileClient.defaultMobileClient().getIdentityManager().getCredentialsProvider());

                //full path of upload image
                final URL presignedUrl =
s3.generatePresignedUrl(AWSConfiguration.AMAZON_S3_USER_FILES_BUCKET, S3_PREFIX_PUBLIC +
contentItem.getFilePath(),
                        new Date(new Date().getTime() + 60 * 60 * 1000));

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

                uploadedImagePath = urlParts[0];

                sendBtn.callOnClick();

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

            @Override
            public void onProgressUpdate(final String fileName, final boolean isWaiting,
                                         final long bytesCurrent, final long bytesTotal) {
            }

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

            }
        });
    }
```

- Once the image is successfully uploaded, get the presigned URL from the `generatePresignedUrl` method of the class `AmazonS3`.

**Download source code**

## 6. Sending Push Notifications to Chat Room Participants

Our application now allows users to create chat rooms, view all chat rooms on a single screen, and send/receive images. Now we must enable it to send/receive messages. This is where **Amazon SNS** comes in. Amazon's Simple Notification Service, or SNS, is the technology that allows you to enable Push Notifications for your application. In the context of a messaging application like ours, Push Notifications would allow our user to send a message to an identified recipient, and notify said recipient that they have received a new message. To implement Push Notifications for our application, we must first enable it on our app's project on Mobile Hub.

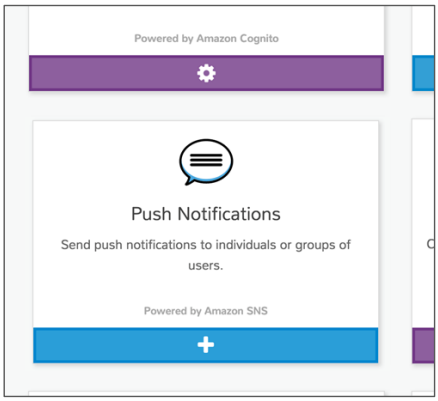
For more information about Amazon's notification service, visit Amazon SNS

**What does it cost?**
Amazon SNS customers can get started with a **free tier**.

Visit SNS Pricing for more information about all the pricing tiers available

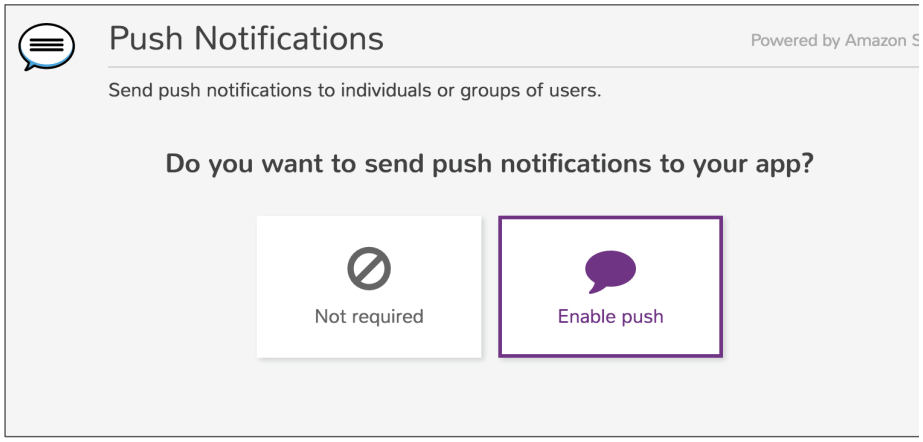
註解 [A12]: Section added

- Go to the project dashboard on Mobile Hub and click on the + button on the "Push Notifications" box.




*Screen 32: Push notification feature*

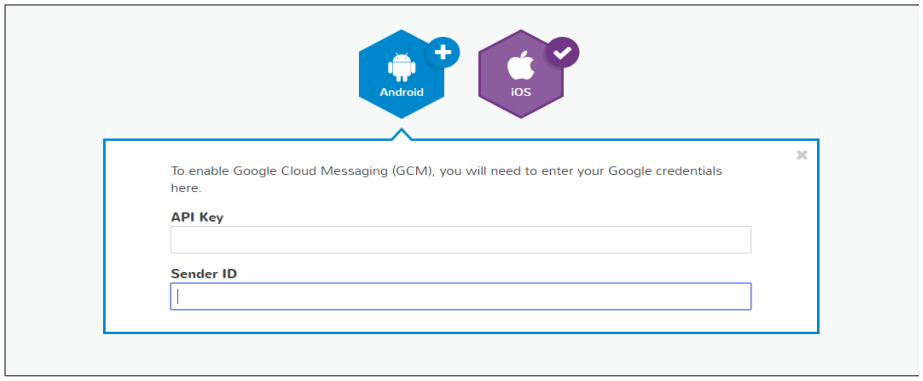
- The next screen would ask you whether you wish to enable push notifications for your app or not. Select "Enable Push"




*Screen 33: Enabling Push Notifications for project*

Once you enable the functionality, you will be required to select one of the two platforms to enable notifications for – Android or iOS.

- Since our app is Android-based, select the Android platform logo




*Screen 34: Platform selection for SNS*

- Put in the API key and Sender ID
- Finally, build the app and run the code

Before we can send out push notifications to a particular recipient (and, as a consequence, the recipient can acknowledge receiving the push notification), we need to identify the endpoint that

we wish to send the notifications to. A single user can use multiple devices, so our app needs to be able to distinguish each device as a different endpoint; hence, we need to store each device's identity separately in the database. Each device has a unique identity, referred to as targetArn. To be able to send out push notifications to a particular device, we need to get its user ID or targetArn.

### How to save user device ID (targetArn)

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

                                Log.i("AWS", msg);

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

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

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

                                                            // create hash set
                                                            Set<String> endpointSet = new
HashSet<String>();

endpointSet.add(pushManager.getEndpointArn());

                                                            String ph =
userInput.getText().toString();
                                                            //add user profile information in
UserProfile table

userProfileManager.addUserProfile(endpointSet, ph ,identityManager);
                                                            startActivity(new
Intent(SignInActivity.this, DashBoardActivity.class)
                                                                    .setFlags(Intent.FLAG_ACTIVITY_C
LEAR_TOP));

                                                            finish();
```

```
                                                dialog.cancel();
                                            }
                                        })
                                .setNegativeButton(R.string.cancel,
                                        new DialogInterface.OnClickListener() {
                                            public void onClick(DialogInterface dialog,
int id) {
                                                dialog.cancel();
                                            }
                                        });

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

                                String phNo =isUser.getItems().get(0).get("phone").getS();
                                Set<String> endpointSet = new HashSet<String>();
                                endpointSet.add(pushManager.getEndpointArn());
                                //add user profile information in UserProfile table
                                userProfileManager.addUserProfile(endpointSet,
phNo ,identityManager);

                                Log.d(LOG_TAG, "Launching ChatRoomHome Activity...");
                                startActivity(new Intent(SignInActivity.this,
DashBoardActivity.class)
                                        .setFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP));
                            }
                        }
                    }
                }.execute();
```

Now that we have the `targetArn`, we can send push notifications from the code.

### Sending Push Notifications from Code

- Get `targetArn` of the recipients
- Set attributes for class `PublishRequest.java`. Attributes include `targetArn` (recipient device ID), `MessageStructure` and `Message`.
- Pass the `PublishRequest.java` object to the `AmazonSNSClient` method `publish`

Following is the code illustrating this:

```
private void sendPush(){

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

                    CognitoCachingCredentialsProvider credentialsProvider = new
CognitoCachingCredentialsProvider(
                            getApplicationContext(),
                            AWSConfiguration.AMAZON_COGNITO_IDENTITY_POOL_ID, // Identity Pool ID
                            Regions.US_EAST_1 // Region
                    );
```

```
                    AmazonSNSClient snsClient = new AmazonSNSClient(credentialsProvider);

                    for (UserProfileDO userData : awsUsersData) {
                        String currentUserID =
identityManager.getCredentialsProvider().getCachedIdentityId();
                        String userID = userData.getUserId();
                        if(!userID.equals(currentUserID)){
                            Set<String> targetArn = userData.getPushTargetArn();
                            // create an iterator
                            Iterator targetArnIterator = targetArn.iterator();
                            for( ;targetArnIterator.hasNext();){
                                PublishRequest publishRequest = new PublishRequest();
                                publishRequest.setMessageStructure("json");
                                publishRequest.setTargetArn(targetArnIterator.next().toString());
                                String sender = identityManager.getUserName();
                                String defaultMessage = String.format("\"default\": \"Sent
By.\",\n",sender);
                                String gcmMessage =
String.format("\"GCM\":\"{\\\"data\\\":{\\\"message\\\":\\\"Message sent by
%s\\\",\\\"chatRoomId\\\":\\\"%s\\\"}}\",\n",sender,chatRoomId);
                                String apnsMessage =
String.format("\"APNS\":\"{\\\"aps\\\":{\\\"alert\\\":\\\"Message sent by
%s\\\"},\\\"chatRoomId\\\":\\\"%s\\\"}\"\n",sender,chatRoomId);
                                String message = String.format("{\n"+ defaultMessage+ gcmMessage +
apnsMessage+"}");
                                publishRequest.setMessage(message);
                                PublishResult publishResult = snsClient.publish(publishRequest);
                                Log.i("AWS", publishResult.getMessageId());
                            }
                        }
                    }
                } catch (AmazonServiceException ex) {
                    msg = ex.getLocalizedMessage();
                    Log.e("CustomError", "---> " + ex.getLocalizedMessage());
                }
                return msg;
            }

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

Once we have enabled the application to send push notifications, we need to allow it to receive push notifications [at the receiving end] as well.

### Receiving Push Notifications from Code

- Get `chatRoomId` from the data sent to receiving device

- Load conversation

Following is the code illustrating the process when the chat activity is open.

```
private final BroadcastReceiver notificationReceiver = new BroadcastReceiver() {
```

```
        @Override
        public void onReceive(Context context, Intent intent) {

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

- Following is the code illustrating the process when the application is running in the background. The code shows the receiving device getting an alert notification.

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

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

        }
    };
```

## Conclusion

In this tutorial we have shown you how you can use multiple AWS services to quickly and easily build an Android-based messaging application. AWS technologies and services used in this tutorial included:

- AWS Mobile Hub
- Amazon Cognito Identity
- Amazon Mobile Analytics
- Amazon DynamoDB
- Amazon S3
- Amazon SNS

With the information of this tutorial in hand, you can begin to envision much richer and advance mobile experiences based on AWS services.

.