

Extend an Android app using the IBM Push for Bluemix cloud service

Use Push notifications to send relevant content to mobile devices

[Belinda Vennam](#)

Software Engineer, IBM Mobile Cloud Services
IBM

21 August 2014

(First published 26 February 2014)

[Salim Zeitouni](#)

Software Engineer, IBM Application and Integration
Middleware
IBM

[Paul Mariduená \(https://www.ibm.com/developerworks/community/profiles/html/profileView.do?key=1f068cb1-8ce6-4b1d-bbdd-b832ec6f436e&lang=en&tabid=dwAboutMe\)](https://www.ibm.com/developerworks/community/profiles/html/profileView.do?key=1f068cb1-8ce6-4b1d-bbdd-b832ec6f436e&lang=en&tabid=dwAboutMe)

Software Engineer, IBM Application and Integration
Middleware
IBM

Build an Android application using the IBM Push for Bluemix and Node.js services available on IBM Bluemix. This multi-part series will gradually build, with each part introducing you to new services. This tutorial includes sample code and complete instructions for creating the BlueList Android application. You can apply what you've learned to integrate the IBM Mobile Data for Bluemix, Node.js, and IBM Push for Bluemix services into your own applications.

You may already know about some of the benefits of [Bluemix](#), IBM's open platform for developing and deploying mobile and web applications. The many pre-built services in the [Mobile solution in Bluemix](#) make it easy to build and enhance applications.

[Tutorials in this series](#) show you how to add cloud-based services to your apps. In this tutorial, we'll extend an Android application using the [IBM Push for Bluemix](#) service.

Looking for the new **iOS 8 beta services in the MobileFirst Platform**? Here's how to [get started](#). And please view our [updated BlueList sample code](#).

READ: [Getting started with Push](#)

As introduced in the [previous tutorial in this series](#), the BlueList application is a simple Android app. In this tutorial, we'll add the IBM Push for Bluemix (invoked from a Node.js-hosted application) to the BlueList app, so that notifications are sent when a list is updated, and the list is updated on all devices when one of the devices updates the list in some way.

“ With the free services on Bluemix, you can get started writing and extending your own applications in no time! ”

[Get bluelist-mobiledata \(v1\) code](#)

[Get bluelist-push \(v2\) code](#)

In the [previous tutorial in this series](#), we added [IBM Mobile Data for Bluemix](#) (Mobile Data), where items in a list persist. We'll now add the capability of sending Push notifications from a Node.js-hosted Cloud application whenever the list is updated. If you skipped the previous tutorial in this series, you can start with the bluelist-mobiledata(v1) code, but you will need to perform the "[Catch up](#)" tasks, before walking through the steps required to add IBM Push for Bluemix (Push) and the Node.js Cloud-hosted application. If you worked through the previous tutorial in this series, you can start with your existing code and follow the steps below to add Push and the Node.js Cloud application. If you want to jump ahead by downloading the bluelist-push(v2) code, you'll need to perform just a few steps to get the BlueList app with the Mobile Data and Push services working with the Node.js Cloud application. The bluelist-push(v2) code includes the changes that are made to the bluelist-mobiledata(v1) code in this tutorial.

What you'll need for your application

- Familiarity with the [previous tutorial in this series](#), where we added the IBM Mobile Data for Bluemix service to an Android app. That app is the starting point for this tutorial.
- Familiarity with [Android development](#).
- An Android development environment. We used Eclipse with ADT, but feel free to use your preference.

- Note:** If you decide to use Android Studio, you can skip the steps [Add Google Play Services to your project](#) and [Add IBM Push and IBM CloudCode Client SDK JARs to your project](#) since the `gradle.build` file provided in the Android root directory will pull in the required dependencies. Please check it out, familiarize yourself with the Gradle build code, and uncomment the required dependencies. You will also need to [download the Gradle.zip](#) and extract the contents to the directory of your choice. When Android Studio prompts you for a `GRADLE_HOME`, use the path you extracted the .zip to, where the bin directory lives.
- Familiarity with [git](#) to download the bluelist-mobiledata(v1) and Node.js application code.
 - bluelist-mobiledata(v1) code (click the button above) or your existing code from the previous tutorial in the series.
 - The Node.js application is in the [bluelist-push-node folder](#) of the bluelist-push project.
 - A GCM Sender ID and API Key from Google. Read how to get this key in [Google's instructions](#) or our instructions.

Note: You will need a Gmail account to perform this task.

Catch up: If you skipped the previous tutorial in this series

1. If you haven't already, download the `bluelist-mobiledata(v1)` code (click the button above).
2. See **Step 1. Create a Mobile Cloud application on Bluemix** in the [previous tutorial in this series](#). Perform steps 1.1 through 1.10. These steps will walk you through logging into Bluemix and creating an application (BlueList) on Bluemix, downloading the Android SDK, updating the `bluelist.properties` file with your Application ID, and adding the `ibmbaas.jar`, `ibmdata.jar`, and `ibmfilesync.jar` to your `libs` directory.
3. Before you add the IBM Push for Bluemix service, run the code in your emulator to verify that your Android development environment is set up correctly. You should be able to perform **Step 7. Run the app** and **Step 8. See your data in the cloud** from the previous tutorial in this series.
If the application runs successfully, you are all caught up and ready to add Push notifications and Node.js Cloud-hosted applications!

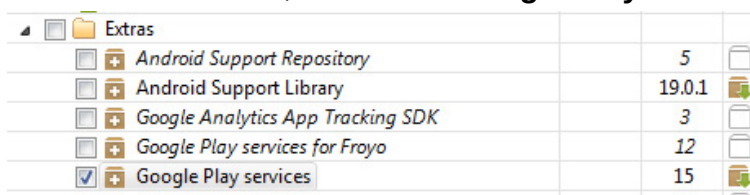
Before you start: Get your Google API Project Number and GCM API Key needed for Push

Get your Google API Project Number and GCM API Key

1. Open the [Google Developers Console](#).
2. Click **CREATE PROJECT**, enter a Project name, click **Create**.
3. Copy the Project Number from the top of the page. This is your GCM Sender Id (Google API Project Number). You'll need this later!
4. Click **APIs & auth** on the left of the page.
5. Turn ON **Google Cloud Messaging for Android**.
6. Under APIs & auth, click **Credentials**.
7. Click **CREATE NEW KEY** under the Public API access section.
8. Click **Server key**.
9. Click **Create**.
10. Copy the API key from the Public API access section. This is your Sender Auth Token (GCM API Key). You'll need this later!

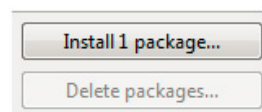
Add Google Play Services to your project (Skip this step if using Android Studio)

1. Open Eclipse, and select **Window > Android SDK Manager**.
2. Scroll to the bottom, and select **Google Play services** under **Extras**:



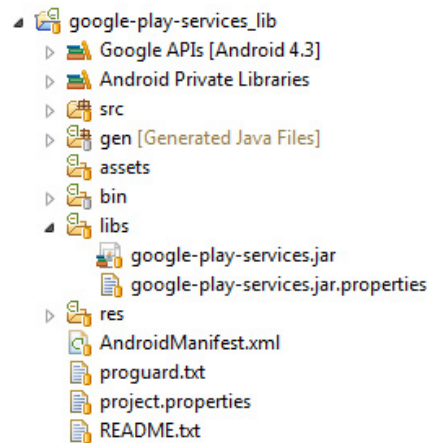
Extras			
<input type="checkbox"/>	Android Support Repository	5	
<input type="checkbox"/>	Android Support Library	19.0.1	
<input type="checkbox"/>	Google Analytics App Tracking SDK	3	
<input type="checkbox"/>	Google Play services for Froyo	12	
<input checked="" type="checkbox"/>	Google Play services	15	

3.



Click **Install 1 package...** and accept the licenses.

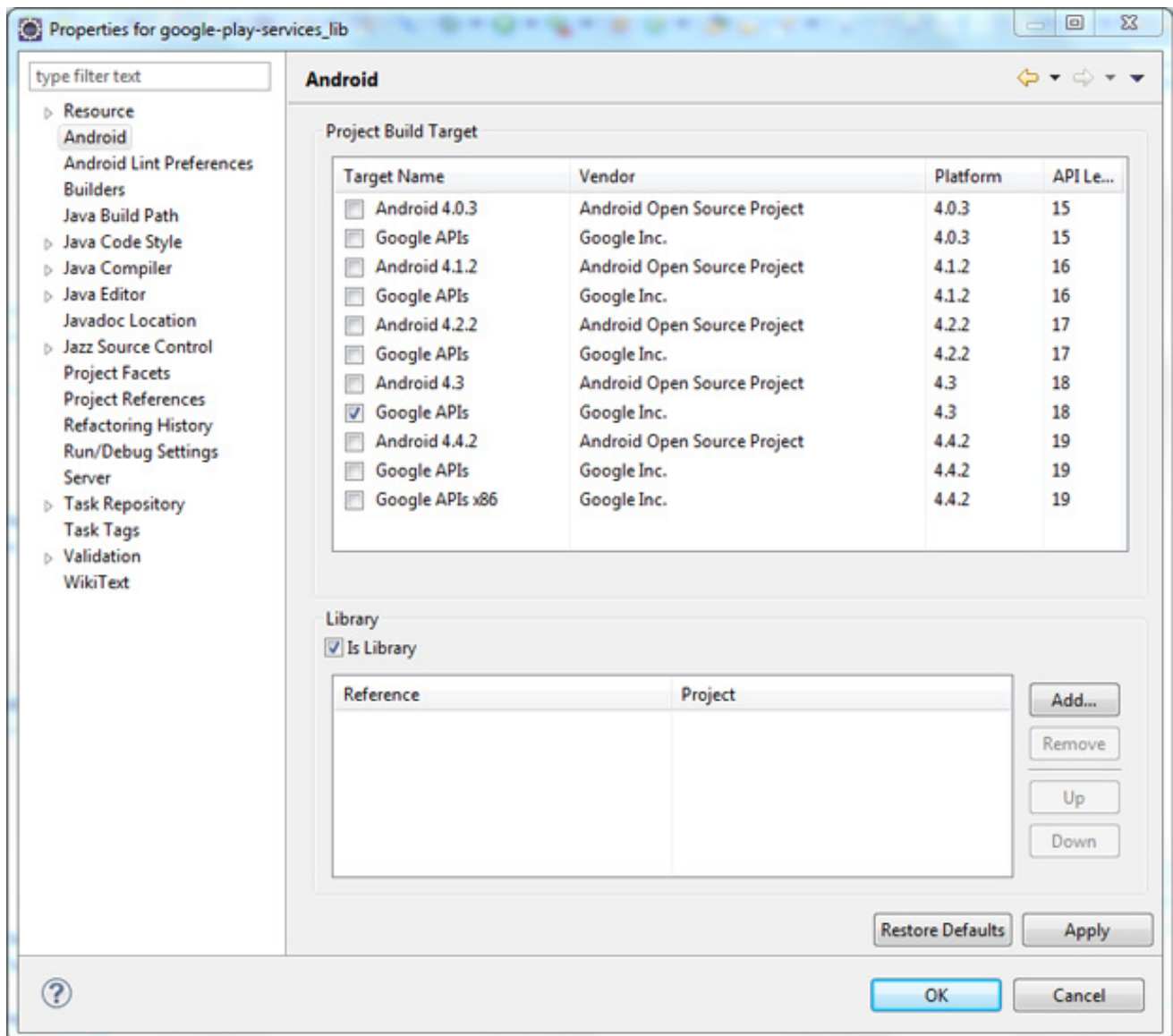
4. After the installation is successful, import the project now located in your file system at `<Android_SDK_Location>\extras\google\google_play_services\libproject\google-play-services_lib` into your Eclipse workspace. To do this, select **File > Import**, and then select **Android > Existing Android Code into Workspace**, and browse to the **google-play-**



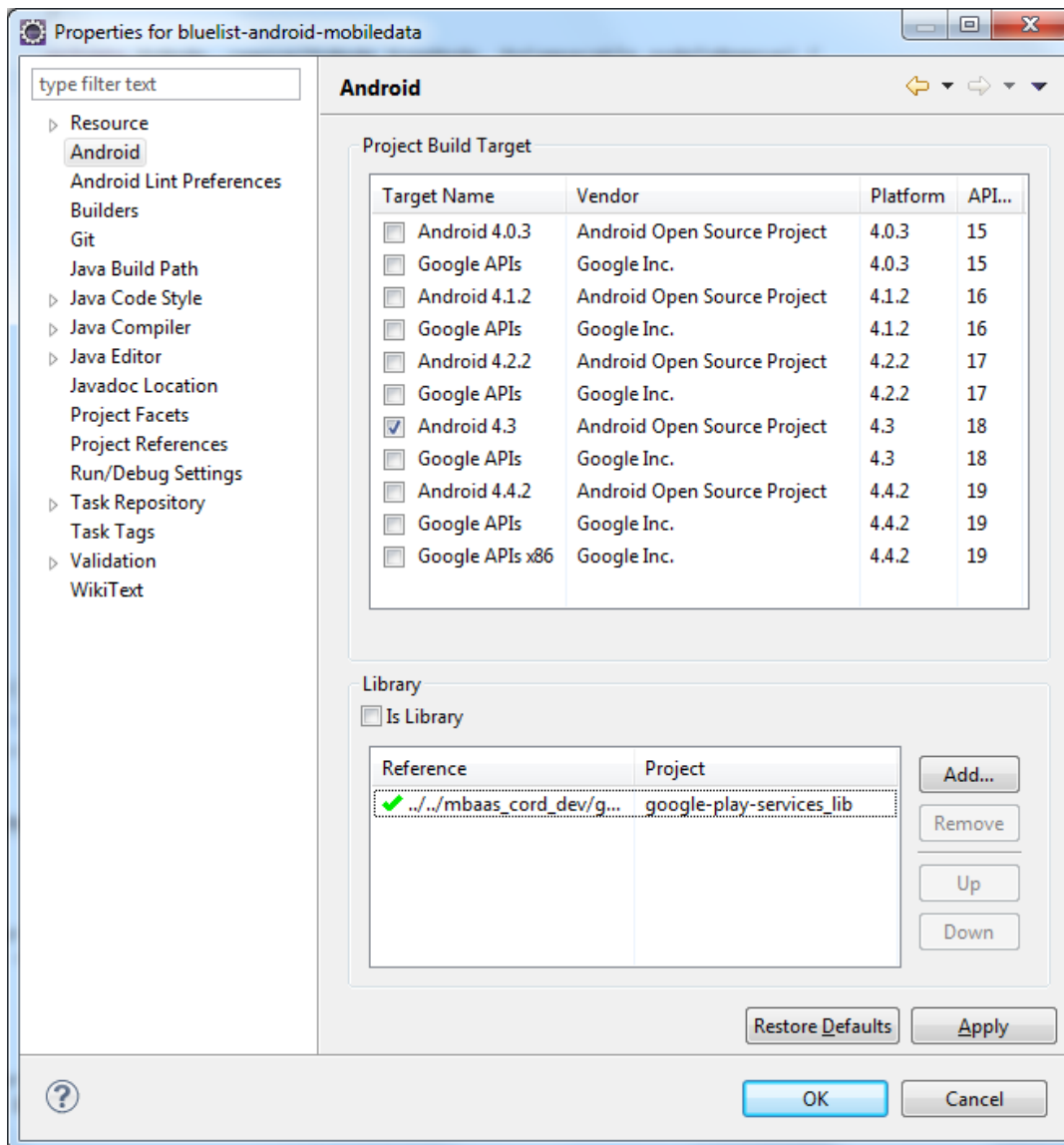
services_lib project.

5. After successfully importing google-play-services_lib into your workspace, it needs to be marked as an **Android library project**. To do this, open

the **Properties** for the project and select the **Is Library** checkbox:



6. Next, your project needs to refer to the new library project. To add the reference to the google-play-services_lib Android library project, follow these steps:
 - a. Make sure that both the project library and the application project that depends on it are in your workspace. If one of the projects is missing, import it into your workspace.
 - b. In the Package Explorer, right-click the dependent project and select **Properties**.
 - c. In the Properties window, select the **Android** properties group at the left, and locate the **Library** property on the right.
 - d. Click **Add** to open the Project Selection dialog.
 - e. From the list of available library projects, select the **google-play-services_lib** project and click **OK**.
 - f. When the dialog closes, click **Apply** in the Properties window.
 - g. Click **OK** to close the Properties window.



7. Add a reference to the google-play-services version at your application's AndroidManifest.xml as the first child of the `<application>` element.

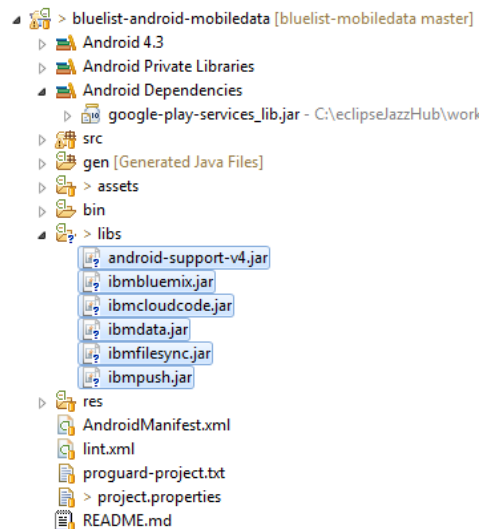
```
<meta-data
    android:name="com.google.android.gms.version"
    android:value="@integer/google_play_services_version" />
```

8. This will result in a compile error, which you can correct by copying the file located at google-play-services_lib/res/values/version.xml into your application's res/values directory.
9. Finally, copy the android-support-v4.jar file into your project's libs directory. The JAR archive can be found at `<Android_SDK_Location>/extras/android/support/v4`

Add IBM Push and IBM CloudCode client SDK JARs to your project (skip this step if using Android Studio)

1. Locate the Android SDK you downloaded in Step 1.7 in the [previous tutorial](#).

2. Copy the `ibmcloudcode.jar` and `ibmpush.jar` files into your project's libs directory. You should now have the following 6 jar files in your libs directory (and one under "Android



Dependencies"):

Step 1. Add the Google API Project Number and GCM API Key into your application in Bluemix

1. Log in to [Bluemix](#).
2. Assuming you've completed the steps in the [previous tutorial](#) or the "Catch up" steps above, click your application in the DASHBOARD view.
- 3.



Click the Push service, `Your_app_name:Push`.

4. Under the Configuration tab of the Push service UI, click **EDIT** under Google Cloud Messaging.
5. Fill in the GCM API Key and the Google API Project Number you got earlier from Google, and

Google Cloud Messaging



Sender Auth Token (GCM API Key):

AlzaSyBrFzzW5JRwO_goiyzULbn83jlu_TOAmN8

Sender Id (Google API Project Number):

923067433657

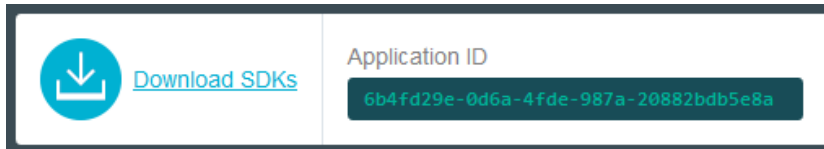
click **SAVE**.

Step 2. Familiarize yourself with server-side Node.js

1. If you haven't already, download the [Node.js application](#). The Node.js application is in the `bluelist-push-node` folder of the `bluelist-push` project.
2. Look through the `bluelist-push-node/app.js` file and familiarize yourself with the code. The `appID` value within the `app.js` code needs to be updated.

Step 3. Import the Node.js application

1. Go to the Overview page for your application, and make note of your **Application ID** for



BlueList.

2. **Use the Cloud Foundry command-line interface:** Download the [Cloud Foundry CLI](#) version 6, and choose the installer appropriate for the system from which you will run the CLI.
3. Open a command prompt, and run the following command to verify proper installation:

```
cf --version
```

If it installed properly, you will see a version returned.

4. Log in to [Bluemix](#) from the CLI by running the command:

```
cf login -a https://api.ng.bluemix.net
```

5. From your command prompt, navigate to the `bluelist-push/bluelist-push-node` directory containing the Node.js application code.
6. List the directory contents to ensure that you see the `app.js`, `manifest.yml`, `package.json`, and the `public` directory.
7. Open `app.js` in a text editor, and add the App ID, and Application Route for your Mobile Cloud application on IBM Bluemix.

```
...
//configuration for application
var appConfig = {
  applicationId: "<INSERT_YOUR_APPLICATION_ID_HERE>",
  applicationRoute: "BlueList.mybluemix.net"
};
....
```

8. Push (upload) the `bluelist-push-node` application up to the Bluemix Node.js runtime by running the following command:

```
cf push BlueList -p .
```

If a 400 error is returned, the host "BlueList" is taken. Update the `host` property in the `manifest.yml` file with a unique name.

9. Confirm that the following success message is returned in the command prompt:


```

1 of 1 instances running

App started

Showing health and status for app BlueList in org <your_org_name> / space <your_space> as
yourid@ibm.com...
OK

requested state: started
instances: 1/1
usage: 512M x 1 instances
urls: BlueList.mybluemix.net

    state      since                cpu    memory          disk
#0  running   2014-04-25 02:23:27 PM  0.0%   53.2M of 512M   34.7M of 1G

```

10. The Node.js application should now be running.

Step 4. Make some client-side code changes to interact with the IBM Push for Bluemix and the Node.js services you just set up

1. Assuming you've completed the steps in the [previous tutorial](#), or in the "Catch up" section above, we're going to start making changes to this project code to bring it up to the bluelist-push(v2) level.
2. If not already done, copy the applicationId, applicationSecret, and applicationRoute values available on your application's Overview page (on Bluemix) to the bluelist.properties file in the assets folder of your project.

```

applicationID=<INSERT_APPLICATION_ID_HERE>
applicationSecret=<INSERT_APPLICATION_SECRET_HERE>
applicationRoute=<INSERT_APPLICATION_ROUTE_HERE>

```

3. Open AndroidManifest.xml, and update the permission and activity sections as shown below. Add the new GCM Intent Service and intent-filters for `RECEIVE` and `REGISTRATION` after the last activity section.

```

<!-- Push Permission -->
<permission
  android:name="com.ibm.bluelist.permission.C2D_MESSAGE"
  android:protectionLevel="signature" />

<!-- Permissions -->
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.WAKE_LOCK" />
<uses-permission android:name="android.permission.GET_ACCOUNTS" />
<uses-permission android:name="android.permission.USE_CREDENTIALS" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<!-- Push Permissions -->
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission android:name="com.ibm.bluelist.permission.C2D_MESSAGE" />
<uses-permission android:name="com.google.android.c2dm.permission.RECEIVE" />

```

```

<activity
    android:name="com.ibm.bluelist.MainActivity"
    android:label="@string/app_name"
    android:configChanges="keyboardHidden|orientation">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>

    <!-- Push Settings Start -->
    <!-- Notification Intent -->
    <intent-filter>
        <action android:name="com.ibm.bluelist.BlueList.IBMPushNotification" />
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
    <!-- Push Settings End -->

</activity>

```

```

<activity
    android:name="com.ibm.bluelist.EditActivity"
    android:label="@string/title_activity_edit"
    android:configChanges="keyboardHidden|orientation">
</activity>

<!-- Push Settings Start -->
<!-- Add GCM Intent Service and intent-filters for RECEIVE and REGISTRATION of notifications -->
<activity android:name="com.ibm.mobile.services.push.IBMUIActivity" />
<service android:name="com.ibm.mobile.services.push.IBMPushIntentService" />

<receiver android:name="com.ibm.mobile.services.push.IBMPushBroadcastReceiver"
    android:permission="com.google.android.c2dm.permission.SEND">
    <intent-filter>
        <action android:name="com.google.android.c2dm.intent.RECEIVE" />
        <category android:name="com.ibm.mbaas.push.sdk.client.android.sample" />
    </intent-filter>

    <intent-filter>
        <action android:name="com.google.android.c2dm.intent.REGISTRATION" />
        <category android:name="com.ibm.mbaas.push.sdk.client.android.sample" />
    </intent-filter>
</receiver>
<!-- Push Settings End -->

```

4. Open `BlueListApplication.java`, and get ready to make some changes.
5. Create an `IBMPush` service and `Activity` variable for later use. Create strings for the `deviceAlias` and `consumerID`. Additionally, create variables for the `IBMPush` service, `IBMPushNotificationListener`, and the `Activity`.

```

public final class BlueListApplication extends Application {
    public static final int EDIT_ACTIVITY_RC = 1;
    public static IBMPush push = null;
    private Activity mActivity;
    private static final String deviceAlias = "TargetDevice";
    private static final String consumerID = "MBaaSListApp";
    private static final String CLASS_NAME = BlueListApplication.class.getSimpleName();
    private static final String APP_ID = "applicationID";
    private static final String APP_SECRET = "applicationSecret";
    private static final String APP_ROUTE = "applicationRoute";
    private static final String PROPS_FILE = "bluelist.properties";

    private IBMPushNotificationListener notificationListener = null;
    List<Item> itemList;
}

```

6. Track the activity status within four of the `ActivityLifecycleCallbacks`.

```

public BlueListApplication() {
    registerActivityLifecycleCallbacks(new ActivityLifecycleCallbacks() {
        @Override
        public void onActivityCreated(Activity activity, Bundle savedInstanceState) {
            Log.d(CLASS_NAME, "Activity created: " + activity.getLocalClassName());
            //Track activity
            mActivity = activity;
        }
        @Override
        public void onActivityStarted(Activity activity) {
            Log.d(CLASS_NAME, "Activity started: " + activity.getLocalClassName());
            //Track activity
            mActivity = activity;
        }
        @Override
        public void onActivityResumed(Activity activity) {
            Log.d(CLASS_NAME, "Activity resumed: " + activity.getLocalClassName());
            //Track activity
            mActivity = activity;
        }
        @Override
        public void onActivitySaveInstanceState(Activity activity, Bundle outState) {
            Log.d(CLASS_NAME, "Activity saved instance state: "
                + activity.getLocalClassName());
        }
        @Override
        public void onActivityPaused(Activity activity) {
            Log.d(CLASS_NAME, "Activity paused: " + activity.getLocalClassName());
            //Track activity
            if (activity != null && activity.equals(mActivity))
                mActivity = null;
        }
        @Override
        public void onActivityStopped(Activity activity) {
            Log.d(CLASS_NAME, "Activity stopped: " + activity.getLocalClassName());
        }
        @Override
        public void onActivityDestroyed(Activity activity) {
            Log.d(CLASS_NAME, "Activity destroyed: " + activity.getLocalClassName());
        }
    });
}

```

7. When the application is created, initialize the IBM Push for Bluemix service and register the device with the service. Use the `continueWith()` method to check the results. You are adding to the existing service initialization calls here. Your updated `oncreate` method should look like this:

```

@Override
public void onCreate() {
    super.onCreate();
    itemList = new ArrayList<Item>();
    // Read from properties file.
    Properties props = new java.util.Properties();
    Context context = getApplicationContext();
    try {
        AssetManager assetManager = context.getAssets();
        props.load(assetManager.open(PROPS_FILE));
        Log.i(CLASS_NAME, "Found configuration file: " + PROPS_FILE);
    } catch (FileNotFoundException e) {
        Log.e(CLASS_NAME, "The bluelist.properties file was not found.", e);
    } catch (IOException e) {
        Log.e(CLASS_NAME, "The bluelist.properties file could not be read properly.", e);
    }
    Log.i(CLASS_NAME, "Application ID is: " + props.getProperty(APP_ID));

    // Initialize the IBM core backend-as-a-service.
}

```

```

IBMBluemix.initialize(this, props.getProperty(APP_ID), props.getProperty(APP_SECRET),
props.getProperty(APP_ROUTE));
// Initialize the IBM Data Service.
IBMDData.initializeService();
// Register Item Specialization here.
Item.registerSpecialization(Item.class);
// Initialize IBM Push service.
IBMPush.initializeService();
// Retrieve instance of the IBM Push service.
push = IBMPush.getService();
// Register the device with the IBM Push service.
push.register(deviceAlias, consumerID).continueWith(new Continuation<String, Void>() {

    @Override
    public Void then(Task<String> task) throws Exception {
        if (task.isCancelled()) {
            Log.e(CLASS_NAME, "Exception : Task " + task.toString() + " was cancelled.");
        } else if (task.isFaulted()) {
            Log.e(CLASS_NAME, "Exception : " + task.getError().getMessage());
        } else {
            Log.d(CLASS_NAME, "Device Successfully Registered");
        }
        return null;
    }
});
}

```

8. When an activity is created, define the `IBMPushNotificationListener`. Your new `onActivityCreated` method should look like this:

```

public void onActivityCreated(Activity activity, Bundle savedInstanceState) {
    Log.d(CLASS_NAME, "Activity created: " + activity.getLocalClassName());
    mActivity = activity;

    //Define IBMPushNotificationListener behavior on push notifications
    notificationListener = new IBMPushNotificationListener() {
        @Override
        public void onReceive(final IBMSimplePushNotification message) {
            mActivity.runOnUiThread(new Runnable() {
                @Override
                public void run() {
                    Class<? extends Activity> actClass = mActivity.getClass();
                    if (actClass == MainActivity.class) {
                        ((MainActivity)mActivity).listItems();
                        Log.e(CLASS_NAME, "Notification message received: " + message.toString());
                        //present the message when sent from Push notification console.
                        if(!message.getAlert().contains("ItemList was updated")){
                            mActivity.runOnUiThread(new Runnable() {
                                public void run() {
                                    new AlertDialog.Builder(mActivity)
                                        .setTitle("Push notification received")
                                        .setMessage(message.getAlert())
                                        .setPositiveButton(android.R.string.ok, new DialogInterface.OnClickListener() {
                                            public void onClick(DialogInterface dialog, int whichButton) {
                                                }
                                            }
                                        ).show();
                                }
                            });
                        }
                    }
                }
            });
        }
    };
}

```

9. Check for push when the activity is resumed or paused. Your new `onActivityResult` method and `onActivityResultPaused` method should look like this:

```
public void onActivityResult(Activity activity) {
    Log.d(CLASS_NAME, "Activity resumed: " + activity.getLocalClassName());
    //Track activity
    mActivity = activity;
    if (push != null) {
        push.listen(notificationListener);
    }
}
```

```
public void onActivityResultPaused(Activity activity) {
    if (push != null) {
        push.hold();
    }
    Log.d(CLASS_NAME, "Activity paused: " + activity.getLocalClassName());
    if (activity != null && activity.equals(mActivity))
        mActivity = null;
}
```

10. In `MainActivity.java`, add a call to `updateOtherDevices` on edit, create, and delete of an item. Add the new `updateOtherDevices()` method, which will make a call to the Cloud-hosted Node.js application (`app.js`) that you previously uploaded to your application.

```
/**
 * Send a notification to all devices whenever the BlueList is modified (create, update, or delete)
 */
private void updateOtherDevices() {

    // initialize and retrieve an instance of the IBM CloudCode service
    IBMCloudCode.initializeService();
    IBMCloudCode myCloudCodeService = IBMCloudCode.getService();
    JSONObject jsonObj = new JSONObject();
    try {
        jsonObj.put("key1", "value1");
    } catch (JSONException e) {
        e.printStackTrace();
    }

    // Call the node.js application hosted in the IBM Cloud Code service
    // with a POST call, passing in a non-essential JSONObject
    // The URI is relative to, appended to, the BlueMix context root

    myCloudCodeService.post("notifyOtherDevices", jsonObj).continueWith(new Continuation<IBMHttpResponse,
    Void>() {

        @Override
        public Void then(Task<IBMHttpResponse> task) throws Exception {
            if (task.isCancelled()) {
                Log.e(CLASS_NAME, "Exception : Task" + task.isCancelled() + "was cancelled.");
            } else if (task.isFaulted()) {
                Log.e(CLASS_NAME, "Exception : " + task.getError().getMessage());
            } else {
                InputStream is = task.getResult().getInputStream();
                try {
                    BufferedReader in = new BufferedReader(new InputStreamReader(is));
                    String responseString = "";
                    String myString = "";
                    while ((myString = in.readLine()) != null)
                        responseString += myString;

                    in.close();
                    Log.i(CLASS_NAME, "Response Body: " + responseString);
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }
    });
}
```

```

        }

        Log.i(CLASS_NAME, "Response Status from notifyOtherDevices: " +
task.getResult().getHttpStatusCode());
    }
    return null;
}
});
}
}

```

```

/*if an edit has been made, notify that the data set has changed.*/
case BlueListApplication.EDIT_ACTIVITY_RC:
// Call updateOtherDevices to invoke Cloud functions
updateOtherDevices();
sortItems(itemList);
lvArrayAdapter.notifyDataSetChanged();
break;
}

```

```

public void createItem(View v) {
    . . .

    // If the result succeeds, load the list.
    else {
        listItems();
        updateOtherDevices();
    }

    . . .
}

```

```

public void deleteItem(Item item) {
    . . .

    // If the result succeeds, reload the list.
    else {
        updateOtherDevices();
        lvArrayAdapter.notifyDataSetChanged();
    }

    . . .
}

```

Step 5. Run the app

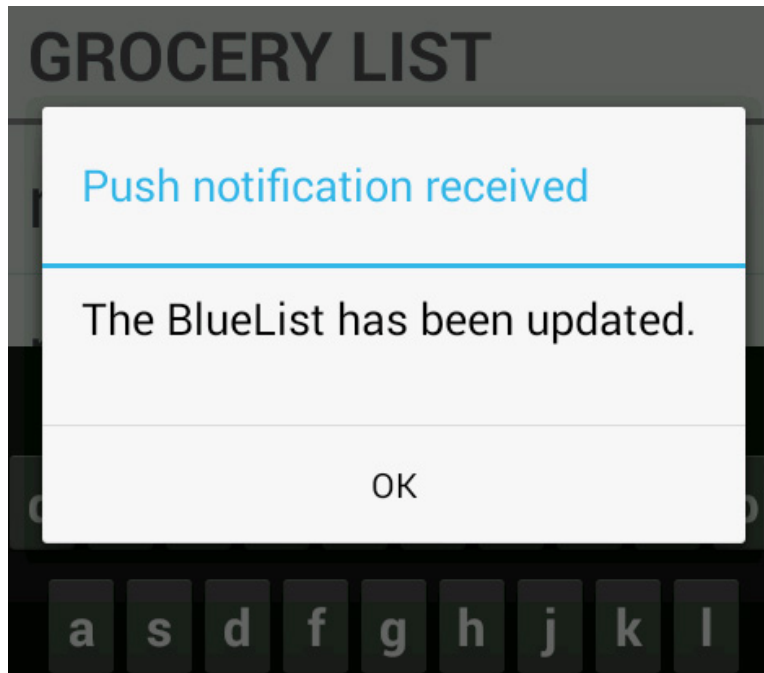
1. Now that you've made the code changes, your code should now be equivalent to version 2!
2. Run the up-to-date code in two separate emulators.

Note: Be sure your Android device supports Google APIs.

- In Eclipse, go to **Project > Properties > Android**.
- Make sure Google APIs for your API level is selected, and click **Apply**.
- Open the Android Virtual Device Manager, select your device, click **Edit**, and set the Target to **Google APIs (Google Inc.) - API Level XX** (where XX matches your API level).

3. Add some grocery list items.
4. Watch as each list gets updated!

5. Additionally, whenever you create, delete, or update an item on the BlueList, all devices registered with the Push service will receive a push notification sent from your Node.js hosted application!



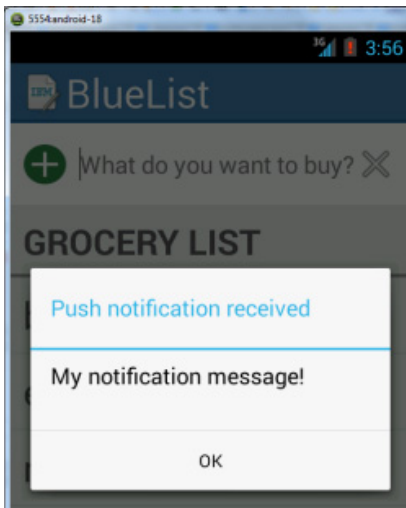
Step 6. Test sending notifications from Bluemix

1. Log in to [Bluemix](#).
2. From the DASHBOARD view, click your application.
3. Click the **Push** service icon.
4. Click the **Notification** tab.
5. Fill in the Message text field with anything you want, and click **NEXT**.

Configuration	Notification	Registrations	Tags
Send notifications to registered mobile devices.			
Compose a Message Step 1 of 2			
*Message text:		<input type="text" value="My notification message!"/>	

6. Choose the Recipients, and click **SEND**.

7. Watch as your mobile device or emulator receives a push notification!



Conclusion

Extending your app using the Push and Node.js server-side services available in [Bluemix](#) should give you a sense of how easy it is to consume and integrate mobile data capabilities using [Mobile services in Bluemix](#). So far in this series, you've used the [IBM Mobile Data for Bluemix](#) service to store, delete, update, and query a list of objects stored on the cloud. You've also used the [IBM Push for Bluemix](#) service and a Node.js Cloud-hosted application to refresh the list on all devices and send notifications when one of the devices updates the list in some way.

The Mobile Data service <http://www.ibm.com/developerworks/topics/mobile%20data%20service> provides easy-to-use SDKs that give access to a scalable, fully managed database via familiar object-oriented APIs. The Push service <http://www.ibm.com/developerworks/topics/push> helps you send relevant content to the right people at the right place and time. The SDK for Node.js runtime <http://www.ibm.com/developerworks/topics/sdk> for node.js runtime helps you develop, deploy, and scale server-side JavaScript apps with ease.

RELATED TOPICS: Android Push notification Node.js

About the authors

Belinda Vennam



[@BeeMarieV](#)

Salim Zeitouni



[Find me on LinkedIn](#)

Paul Mariduená



[Find me on LinkedIn](#)

© Copyright IBM Corporation 2014

(www.ibm.com/legal/copytrade.shtml)

[Trademarks](#)

(www.ibm.com/developerworks/ibm/trademarks/)