Progress®

Developer Guide: Setting Up Push Notifications using Kinvey BaaS

OpenEdge®

1

Introduction

You can develop a mobile app for your OpenEdge application using the Kinvey BaaS platform. Kinvey BaaS is a mobile Backend as a Service (BaaS) platform that provides a rich set of features on the cloud such as user management, data storage, location services, and push notifications, that take away a lot of the complexity involved in developing a mobile app.

This guide focuses on the push notification feature in Kinvey and describes how to set up push notifications that are sent from an OpenEdge application to a mobile app through Kinvey.

Currently, Kinvey supports push notifications for the following types of mobile apps:

- Native Android apps developed using the Android SDK
- Native iOS apps developed using the iOS SDK
- Android or iOS apps developed using NativeScript
- Hybrid apps developed using Adobe PhoneGap

As you proceed through this guide, you will see example code for each type of mobile app, except for hybrid apps developed using PhoneGap. To follow the examples in this guide, you need to be familiar with OpenEdge ABL development, Android or iOS app development, and NativeScript development (if you are developing a mobile app in NativeScript). If you are interested in setting up push notifications for a PhoneGap app, visit the Kinvey PhoneGap push notifications guide.

Note: If you are migrating your mobile app from Telerik Platform to Kinvey BaaS, you may find it useful to first look at the Telerik Platform Migration Guide. To learn about the differences between Telerik Platform and Kinvey in sending push notifications, see Push notification differences between Kinvey and Telerik Platform on page 47.

2

How push notifications work

A push notification is a message that is sent by a mobile app publisher to a mobile app user's device. The message is displayed either in the device's notification pane or in the mobile app itself. Examples of push notifications include marketing or promotional messages, chat messages, alerts, and status changes. Before you read about how to set up push notifications from OpenEdge, it is useful to know how push notifications work.

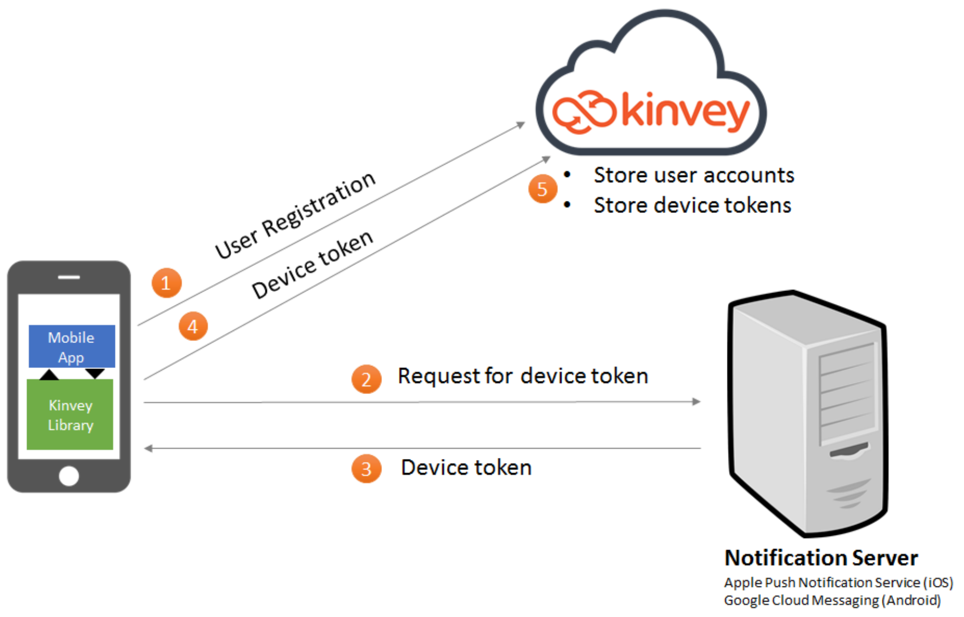
At a high-level, sending push notifications involves two processes:

- **User Registration**, where a user installs the mobile app and gets registered for push notifications with a notification server. Depending on the app code, the registration happens either when the user launches the app or after the user logs in.
- **Push Notification Transmission**, where push notifications are sent from the mobile app publisher to the mobile app user's device through a notification server.

Push notifications using OpenEdge and Kinvey

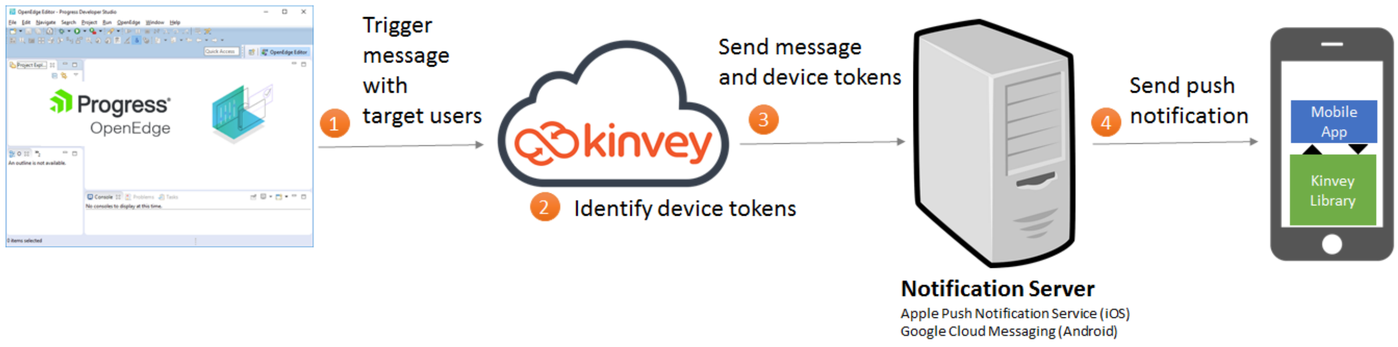
In the case of OpenEdge and Kinvey, your OpenEdge application is responsible for only triggering the push notification message and supplying the message content. User registration and push notification transmission is handled by two Kinvey components--the Kinvey BaaS backend on the cloud and the Kinvey client library (that needs to be added to the app code during development). The details of each of these processes are described below.

User Registration



1. A user installs the mobile app and sets up a user account. The Kinvey client library, which is added to the app, registers the user with the Kinvey BaaS backend.
2. The Kinvey client library requests a token from the notification server. Kinvey uses Apple Push Notification Service (APNS) as the notification server for iOS devices and Google Cloud Messaging (GCM) server for Android.
3. The notification server responds with a device token, which is received by the mobile app.
4. The Kinvey client library sends the device token to the Kinvey backend.
5. The Kinvey backend stores the token and associates it with the user.

Push Notification Transmission


Progress OpenEdge
1 Trigger message with target users
kinvey
2 Identify device tokens
3 Send message and device tokens
Notification Server
Apple Push Notification Service (iOS)
Google Cloud Messaging (Android)
4 Send push notification
Mobile App
Kinvey Library

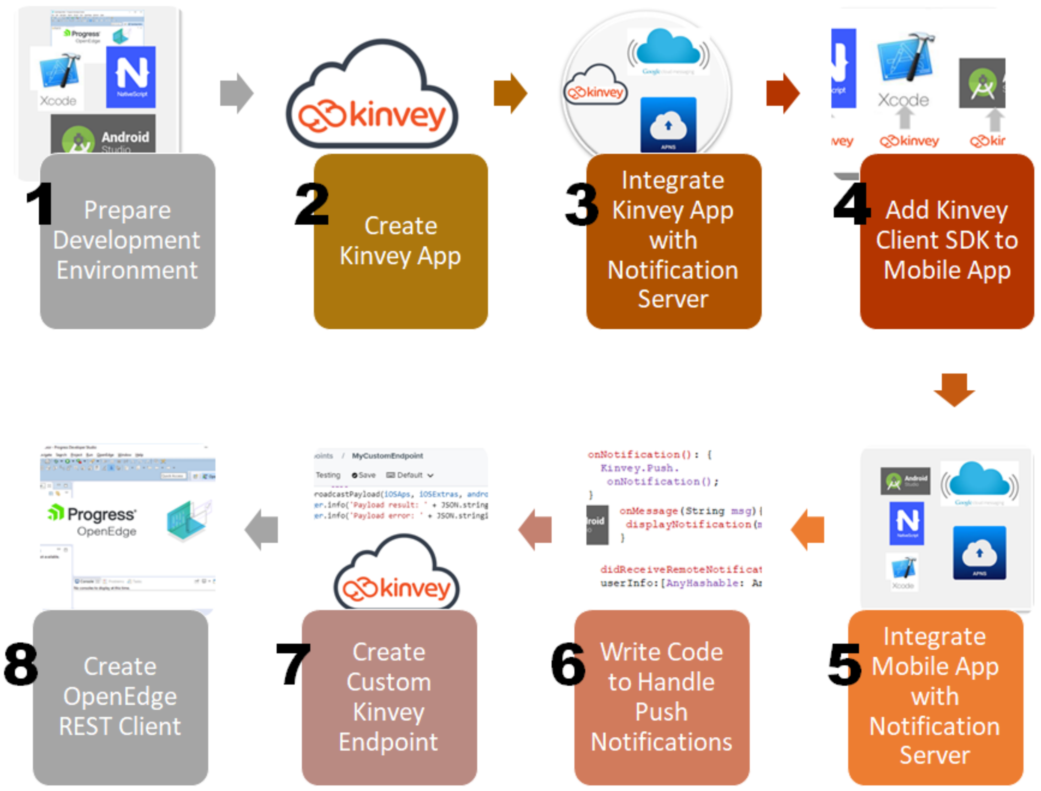
1. An OpenEdge application triggers a push notification by sending a message to an endpoint in the Kinvey backend. The message includes the push notification content that needs to be delivered and information about target users (such as user ID, email, user group, etc).

   Note: The endpoint is a custom endpoint that you need to create in the Kinvey backend. If you want to add any filtering logic, you must implement it in the custom endpoint and use Kinvey's `push` API to send push notifications.

2. The endpoint in the Kinvey backend uses the target users' information to determine each user's device token and forms a push notification message or payload.
3. The Kinvey endpoint sends the push notification to the notification server along with the device tokens.
4. The push notification server sends the push notification to each user.

3

Overview of steps to set up push notifications using Kinvey


1 Prepare Development Environment
2 Create Kinvey App
3 Integrate Kinvey App with Notification Server
4 Add Kinvey Client SDK to Mobile App
5 Integrate Mobile App with Notification Server
6 Write Code to Handle Push Notifications
7 Create Custom Kinvey Endpoint
8 Create OpenEdge REST Client

Perform the following steps to set up push notifications using Kinvey:

1. Prepare your development environment.
2. Create a Kinvey app in the Kinvey backend.
3. Integrate the Kinvey app with the notification server.
4. Add the Kinvey client library to your mobile app.
5. Integrate your mobile app with the notification server.
6. Write code in your mobile app to handle push notifications and register users.
7. Create a custom Kinvey BaaS endpoint.
8. Create an OpenEdge REST client to trigger push notifications.

Note:

This developer guide describes all the steps that are required to set up push notifications using Kinvey. However, the goal of this guide is to enable you to use OpenEdge as the trigger for sending push notifications. If you are interested in generic steps to set up push notifications in Kinvey, you will find them in the Kinvey documentation set:

- Kinvey Android push notification guide
- Kinvey iOS push notification guide
- Kinvey NativeScript push notification guide

Downloading sample projects

The screenshots and sample code that you will see in this guide are based on a simple Login application. If you want to explore the sample OpenEdge, Android, iOS, or NativeScript projects that support the examples in this guide, you can download them from GitHub.

4

Step 1: Prepare your development environment

Before you can set up push notifications, you need to prepare your development environment as follows:

1. Install **OpenEdge Developer Studio** and create an OpenEdge server project. You will need to develop your OpenEdge application as an ABL REST client that triggers push notifications by sending a request message to a Kinvey endpoint. Towards the end of this guide, you will see an example of a basic OpenEdge REST application that triggers a push notification message.
2. Sign up for a Kinvey BaaS account to access the Kinvey backend where you need to create a Kinvey app.
3. Have access to a Google account, if you want to set up push notifications for an Android app (if you have a Gmail account you are all set). The notification server that you will use (Google Cloud Messaging) requires a Google account.
4. Have access to a paid Apple account that is enrolled in the Apple Developer Program, if you want to set up push notifications for an iOS app. The notification server that you will use (Apple Push Notification service) requires an Apple account ID.

In addition, you need to install different development tools depending on the type of mobile app that you are developing (see below). If you are unsure about which approach to take, we recommend trying NativeScript. NativeScript is an open-source framework for developing cross-platform mobile apps using popular technologies like Angular, JavaScript, and TypeScript.

For native Android apps

Perform the following steps for a native Android app:

1. Download and install Android Studio. Supported operating systems include Windows, Mac, and Linux.
2. Create an Android Studio project.
3. Create an Android Virtual Device that will run on an Android Emulator.
4. Add Android activities and develop app code in Java. If you are new to Android development, this is a good place to start.

For native iOS apps

Perform the following steps for a native iOS app:

1. Download and install Xcode. You require a Mac operating system to install Xcode and develop an iOS app.
2. Create an Xcode project.
3. Optionally, set up an iOS Simulator.

   Note: While it is a good practice to use a Simulator to test your iOS app, push notifications do not get displayed in a Simulator. To test push notifications, you will need to install the app in an iPhone.

4. Create storyboards and develop app code in Swift--the programming language for developing iOS apps. If you are new to iOS app development, this is a good place to start.

For NativeScript apps

NativeScript is an open-source framework for developing cross-platform Android and iOS mobile apps using popular technologies like Angular, JavaScript, and TypeScript. Perform the following steps for a NativeScript app:

1. Install a text editor like Visual Studio Code or Atom.
2. Download and install Node.js. Node.js is a framework for developing server-side applications using JavaScript. NativeScript is built on top of Node.js. When you install Node.js, you also get access to **npm** (node package manager), which you use to install NativeScript.
3. Install the NativeScript CLI (Command Line Interface) by following these steps.
4. Write your app code in JavaScript, TypeScript, or Angular. If you are new to NativeScript app development, this is a good place to start.

5

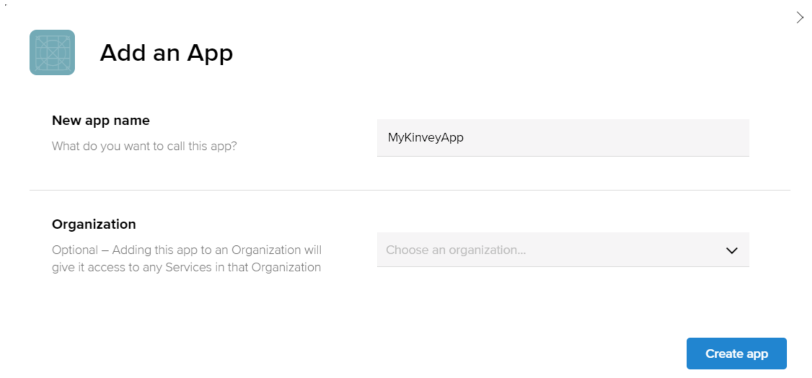
Step 2: Create a Kinvey app

You create a Kinvey app in the Kinvey console--a browser-based development environment that enables you to access and configure Kinvey's cloud-based services. You use the app to store user accounts and write custom business logic to send push notifications to a notification server.

To work with Kinvey, you need to first have a Kinvey account. If you do not have an account, you can create one by signing up. Note that your account gets associated with a free **Developer** plan by default. If you want to browse through all the plans, visit the Pricing page.

If you already have an account, log into the Kinvey console. Then, follow these steps to create a Kinvey app:

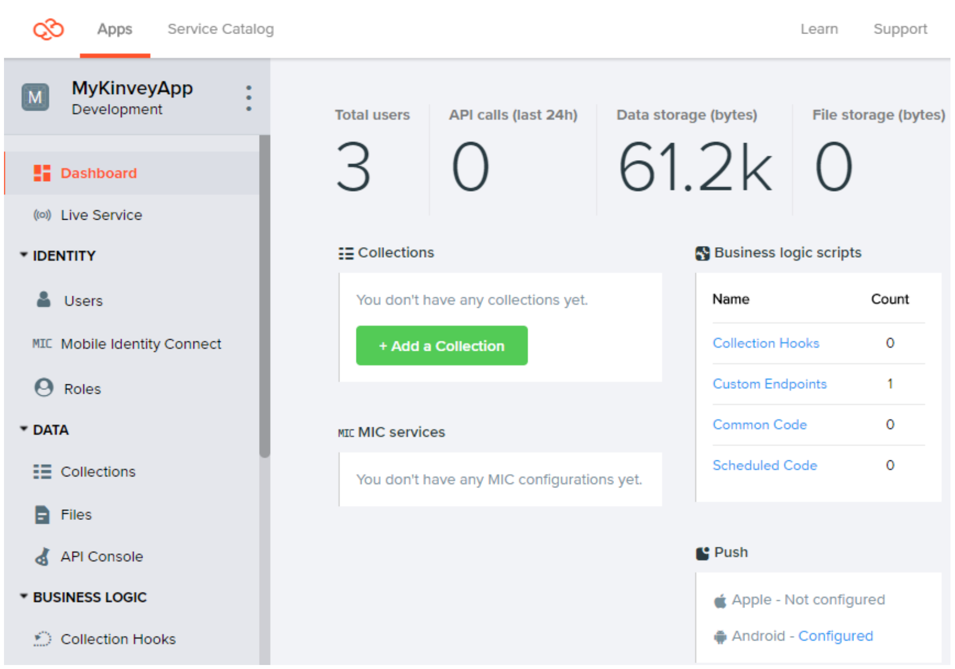
1. In the Kinvey console, click **Add an app**.
2. In the **Add an app** dialog box, enter a name for the app, and optionally, the name of an organization that the app will belong to. Then, click **Create app**.



New app name
What do you want to call this app?
MyKinveyApp

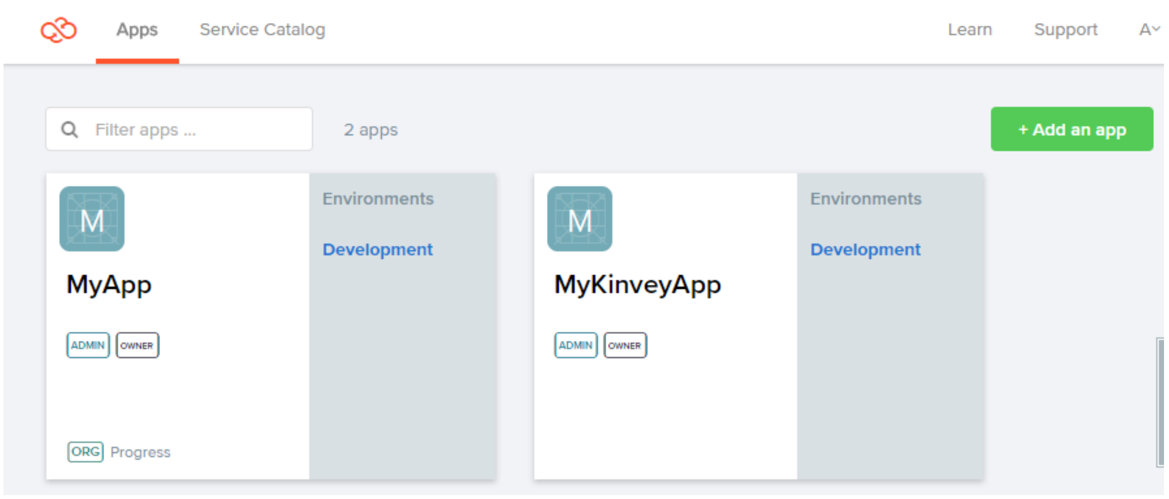
Organization
Optional – Adding this app to an Organization will give it access to any Services in that Organization
Choose an organization...

The app should automatically open and display a page similar to the image below. The left pane gives you access to all of Kinvey's services such as user management, data storage, push notifications, etc.



Name	Count
Collection Hooks	0
Custom Endpoints	1
Common Code	0
Scheduled Code	0

If you happen to reopen the Kinvey console at any point of time, you will see your app name displayed on a card as shown in the following image. To open the app, click **Development**. Note that clicking anywhere else in the card opens the app's **Manage App** page, which provides access to administrator tasks such as upgrading the plan, deleting the app, etc.



6

Step 3: Integrate the Kinvey app with the notification server

You can integrate each Kinvey app that you create with one Android and one iOS notification server. To send push notifications to Android apps, Kinvey uses **Google Cloud Messaging** (GCM). For iOS apps, it uses **Apple Push Notification Service**.

The steps to integrate a Kinvey app with each of these notification servers are explained below. If you are adding push notification support to a NativeScript mobile app, you may need to integrate the Kinvey app with both notification servers, depending on whether you plan on deploying your app to one or both mobile operating systems.

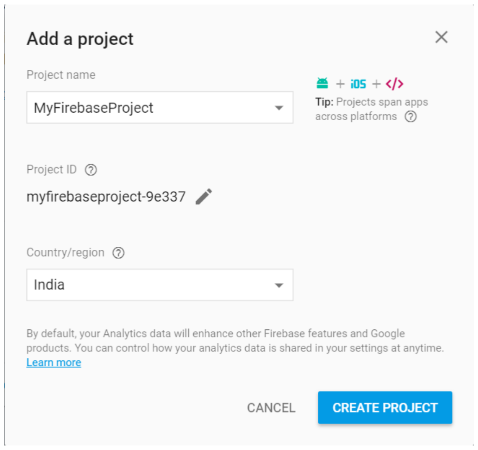
Integrating with Google Cloud Messaging for Android

While Kinvey uses GCM, you need to set up a Firebase Cloud Messaging project and use the project's properties to integrate Kinvey with Google Cloud Messaging. Firebase Cloud Messaging is the updated version of GCM.

Add a Firebase project

Log in to the Firebase console using your Google account. Add a project in Firebase console as follows:

1. Click **Add Project** in the console.
2. In the **Add a project** dialog box, enter a name for the project, select your **Country/region**, and click **CREATE PROJECT**.



Add a project

Project name: MyFirebaseProject

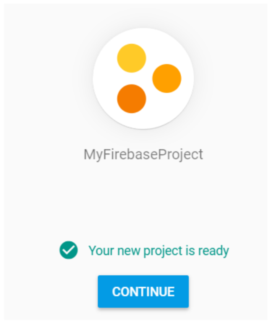
Tip: Projects span apps across platforms

Project ID: myfirebaseproject-9e337

Country/region: India

By default, your Analytics data will enhance other Firebase features and Google products. You can control how your analytics data is shared in your settings at anytime. Learn more

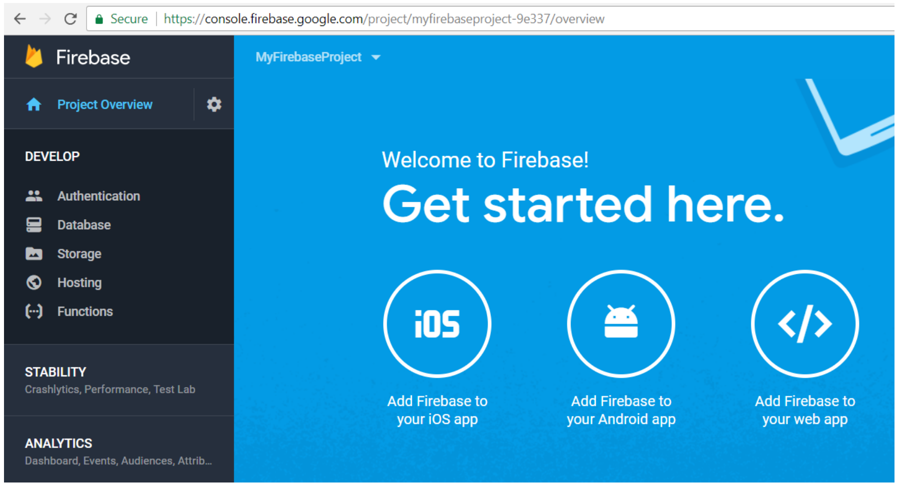
3. Click **CONTINUE**.



MyFirebaseProject

Your new project is ready

You should now be able to see a project home page that looks like this:



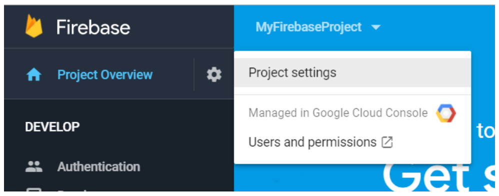
Secure | https://console.firebase.google.com/project/myfirebaseproject-9e337/overview

Firebase
Project Overview
DEVELOP
Authentication
Database
Storage
Hosting
Functions
STABILITY
Crashlytics, Performance, Test Lab
ANALYTICS
Dashboard, Events, Audiences, Attrib...
MyFirebaseProject
Welcome to Firebase!
Get started here.
Add Firebase to your iOS app
Add Firebase to your Android app
Add Firebase to your web app

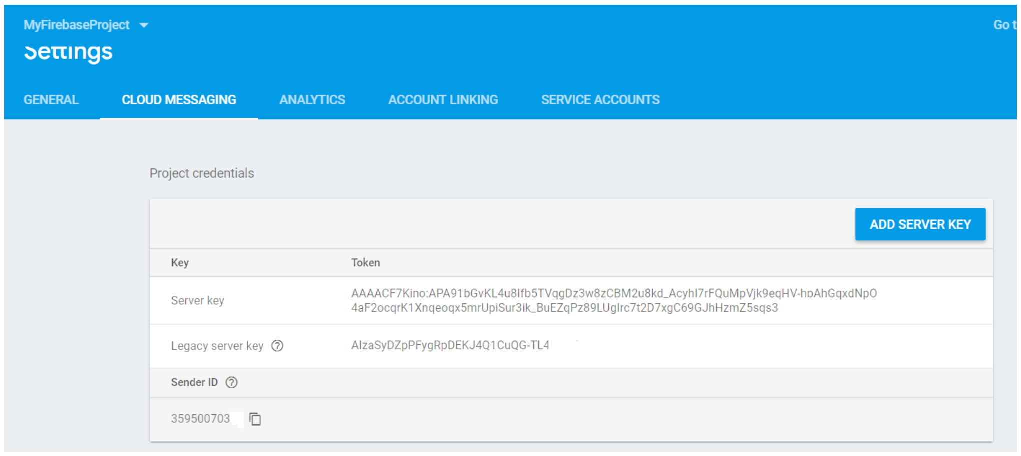
Integrate the Kinvey app with GCM

To enable Kinvey to send push notifications to GCM, you need to configure the Firebase project's sender ID and server key in your Kinvey app.

1. In the Firebase project's home page, click the gear icon next to **Project Overview** and select **Project settings**.



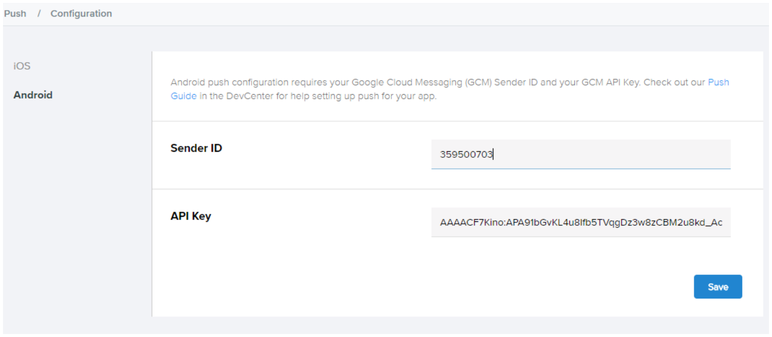
2. Click the **Cloud Messaging** tab and copy the **Sender ID** and **Server Key** and store it in a text file. *TIP:* When you hover over any of these fields, you will see a Copy icon. You can click this Copy icon to copy the value to the clipboard.



Key	Token
Server key	AAAACF7Kino:APA91bGvKL4u8Ifb5TVqgDz3w8zCBM2u8kd_AcyhI7rFQuMpVjk9eqHV-hpAhGqxdNpO 4aF2ocqrK1Xnqeoqx5mrUpiSur3ik_BuEZqPz89LUgIrc7t2D7xgC69GJhHzmZ5sqs3
Legacy server key	AIzaSyDZpPFygRpDEKJ4Q1CuQG-TL4
Sender ID	
359500703	

3. In the Kinvey app, open the app homepage and click **Push** under **ENGAGEMENT**. Then click **Configure Push**.

4. In the **Configuration** page, click **Android**, and then enter the Firebase sender ID and server key. Finally, click **Save**. Your Kinvey app is now integrated with Firebase.



Android push configuration requires your Google Cloud Messaging (GCM) Sender ID and your GCM API Key. Check out our Push Guide in the DevCenter for help setting up push for your app.

Sender ID: 359500703

API Key: AAAACF7Kino:APA91bGvKL4u8Ifb5TVqgDz3w8zCBM2u8kd_Ac

Integrating with Apple Push Notification Service for iOS

To integrate a Kinvey app with Apple Push Notification Service, you need an Apple push certificate file that has the extension **.p12**. The p.12 certificate file contains mainly two things--an SSL public key certificate that is digitally signed by Apple and a private key. Kinvey uses the public key to encrypt push notification messages that it sends to the Apple Push Notification Service. You do not need to write any code to handle this--you can generate the .p12 certificate file using various Apple developer tools and then add it to the Kinvey app.

Apple enables you to generate two types of p.12 certificate files--one for development and another for production. You can use either with Kinvey based on your requirements. However, note that you can use the production certificate only if your app is in the Apple App Store. In the steps listed below, the assumption is that you will use a development certificate.

Perform the following steps to generate an Apple push certificate and add it to the Kinvey app.

Create an App ID

An Apple App ID is a string that identifies an iOS app. Typically, the string consists of a Team ID (generated by Apple) and a bundle ID, with a period (.) separating the two parts. If you are developing a native iOS app, you have two options--you can either create the App ID from your Xcode project or perform the following steps to create it in the Apple developer portal. If you are developing a NativeScript iOS app, you must perform the following steps to create an App ID.

1. Log in to the Apple developer portal. (Remember that you need a paid developer membership to follow the next steps.)
2. Select **Certificates, IDs & Profiles** in the left pane.
3. Click **App IDs** under **Identifiers** and then click the add icon to the top-right.
4. Enter a name for the App ID, select **Explicit App ID** and then enter the bundle ID. For a native iOS project developed in Xcode, you need to specify your project's bundle ID. To see your project's bundle ID, select your target in the project editor in Xcode, click the **General** tab and see the bundle ID under **Identity**. For a NativeScript app created using NativeScript Sidekick, you should specify the App ID that you configured while creating the app. If you did not use NativeScript Sidekick, you can enter any meaningful ID (for example you could follow the convention, com.<myNamespace>.<myAppName>)--you only need to make sure that the bundle ID is unique in the App Store.
5. Select **Push Notifications** under **App Services > Enable Services** and click **Continue**.
6. In the **Confirm your App ID** page, click **Register** at the bottom.
7. Click **Done** in the **Registration complete** page.

Create a certificate signing request file

To get an Apple push notification SSL certificate, you need to generate a certificate signing request file.

1. Launch **Keychain Access.app**, located in **Applications/Utilities**.
2. Select **Keychain Access > Certificate Assistant > Request a Certificate from a Certificate Authority**. The **Certificate Assistant** window opens.
3. In the Certificate Assistant window, enter email address and common name, click **Saved to disk**, select **Let me specify key pair information**, and then click **Continue**.
4. Specify a location to save the file.
5. In the **Key Pair Information** screen, select **Key Size** and **Algorithm** (retain the defaults if you are unsure) and click **Continue**.
6. Click **Done**. The certificate signing request file should get generated in the location that you specified with the extension **.certSigningRequest**.

Get an Apple push notification SSL certificate

The next task is to get an Apple push notification certificate using the certificate signing request file.

1. Open the Apple developer portal again and in the **Certificates, Identifiers, and Profiles** page, select **Development** under **Certificates** in the left pane.
2. Click the add icon in the top right.
3. Choose **Apple Push Notification service SSL (sandbox)** and click **Continue**.
4. Select the **App ID** that you created earlier and click **Continue**.
5. Click **Continue** in the **About Certificate Signing Request** page.
6. In the **Generate your certificate** page, upload the certificate signing request file that you created earlier and click **Continue**.
7. In the **Your certificate is ready** page, click **Download** to download the SSL certificate. After the download completes, you should be able to see the certificate created with the App ID name.

Export the certificate to a .p12 file

1. Expand the certificate that you downloaded, right-click the private key, and select **Export** ***`CommonName`***.
2. Enter a name for the .p12 file in the **Save As** field and click **Save**.
3. Specify and password and click **OK**.

Add the .p12 file to the Kinvey app

1. In the Kinvey app, open the app homepage and click **Push** under **ENGAGEMENT**. Then click **Configure Push**.
2. In the **Configuration** page, click **iOS**, and then drag and drop the .p12 file that you exported earlier. Enter the password for the .p12 file that you specified earlier and select **Development**. Finally, click **Save**. Your Kinvey app is now integrated with Apple Push Notification service.

7

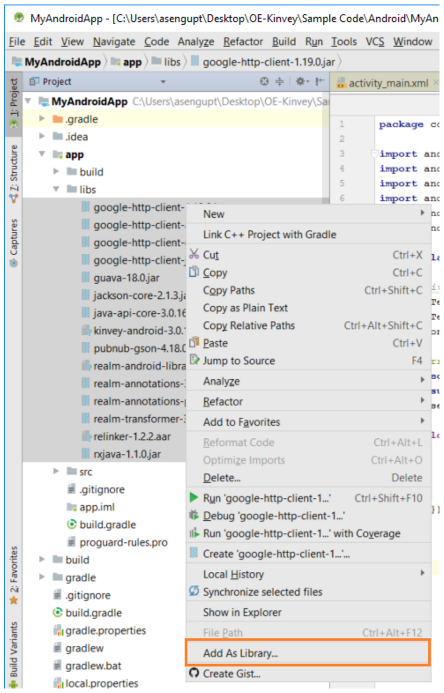
Step 4: Add the Kinvey client library to your mobile app

The Kinvey Client SDK provides services that make it easy for you to do things like registering users with your Kinvey app in the backend, receiving push notifications, etc. Kinvey provides client libraries for each type of mobile app--Android, iOS, and NativeScript.

Add the Kinvey Android client SDK

Follow these steps to add the Kinvey Android Client SDK to your Android app project:

1. Download the Kinvey Android Client SDK. This downloads a zip file containing the client library.
2. Extract the zip file and copy all the files in the **lib** folder of the extracted directory--**kinvey-android-3.0.16**--to the **libs** folder of your Android project (`<AndroidProject>/app/libs`).
3. Right-click the files in the **libs** folder in Android Studio and select **Add As Library**.

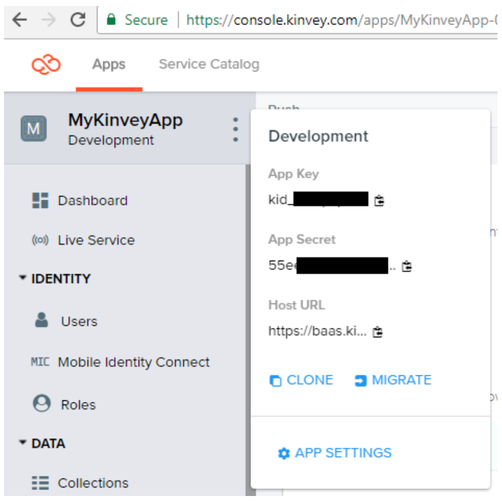


4. Adding the Kinvey SDK files as a library in Android Studio automatically adds dependencies in the **app** module's **build.gradle** file. However, you need to add the following dependencies manually:
   - `implementation fileTree(include: ['*.aar'], dir: 'libs')`
   - `annotationProcessor files('libs/pubnub-gson-4.18.0-all.jar')`
   - `annotationProcessor files('libs/realm-annotations-processor-3.2.1.jar')`
5. You should now see a message prompting you to perform a project sync. Click **Sync Now** and wait for the build to complete.

Add a `kinvey.properties` file

To enable your Android app to communicate with Kinvey, you need to provide your Kinvey app's app key and app secret in a properties file as follows:

1. Create a folder named **assets** in the `app\src\main` directory in your Android app project.
2. In the **assets** folder, create a file named `kinvey.properties`.
3. Define two properties in the `kinvey.properties` file--**app.key** and **app.secret**.
4. Open the home page of your Kinvey app and click the ellipsis next to the app name. Then, copy the **App Key** and the **App Secret** values into the `kinvey.properties` file.



App Key
kid_[redacted]

App Secret
55e[redacted]..

Host URL
https://baas.ki...

Create a Kinvey client

The Kinvey `Client` class enables your app to interact with Kinvey's services from an Android activity. Whether you are registering users or enabling push notifications, you need to use the Kinvey client to access those services.

Initialize the Kinvey client as follows:

1. Import the Kinvey Client class:

   ```
   import com.kinvey.android.Client;
   ```

2. Initialize the Kinvey client in the `onCreate()` method of an Android activity class:

   ```
   Client myKinveyClient = new Client.Builder(this).build();
   ```

   Note: Passing the current context using `this` in `Client.Builder` automatically provides a default reference to the app key and app secret that you defined in the `kinvey.properties` file.

Test the connection with Kinvey

Before you develop your Android app any further, you should test whether it is able to connect to your Kinvey app. The Kinvey client library enables you to test the connection through a service method named `Ping`. Use this method as follows:

```
myKinveyClient.ping(new KinveyPingCallback() {
            @Override
            public void onSuccess(Boolean aBoolean) {
                Log.i(TAG,"Successfully connected to Kinvey");

            }

            @Override
            public void onFailure(Throwable throwable) {
                Log.i(TAG,"Connection to Kinvey failed");

            }
        });
```

Then, run the app in Android Studio. You should be able to see your custom success message in the output log (**logcat**) in Android Studio.

Add the Kinvey iOS client SDK

There are three ways to add the Kinvey iOS client SDK:

- By installing it using CocoaPods. Create a Podfile containing the following statements:

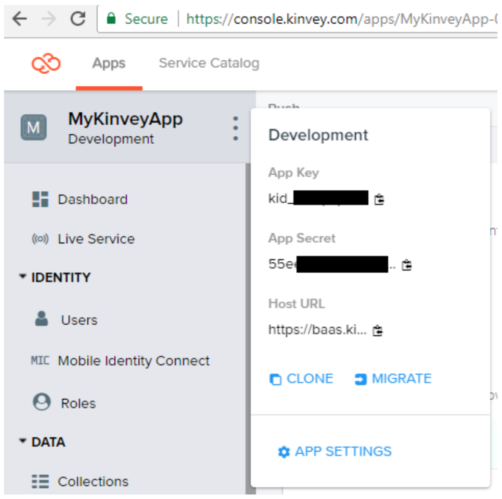
  ```
  target 'MyProjectName' do
    pod 'Kinvey'
  end
  ```

  Then, run `pod install` in the project folder using the Terminal.
- By downloading and importing an iOS starter app containing the Kinvey iOS client SDK.
- By downloading and using the Kinvey iOS SDK's source code. (The SDK is open source).

Create a Kinvey client

After you install the Kinvey iOS client SDK, you can initialize the Kinvey client in your iOS app in Xcode as follows:

1. Open the home page of your Kinvey app and click the ellipsis next to the app name. Then, copy the **App Key** and the **App Secret** values into a text file. You will need these values to initialize the Kinvey client.



2. Initialize the Kinvey client as follows:

```
import Kinvey

 Kinvey.sharedClient.initialize(
     appKey: "<#Your app key#>",
     appSecret: "<#Your app secret#>"
 ) {
     switch $0 {
      case .success(let user):
         if let user = user {
             print("\(user)")
         }
      case .failure(let error):
        print("\(error)")
     }
}
```

Typically, you will want to add this to your AppDelegate's `application:didFinishLaunchingWithOptions` method.

Test the connection with Kinvey

Before you develop your iOS app any further, you should test whether it is able to connect to your Kinvey app. The Kinvey client library enables you to test the connection through a function named `Client.Ping()`. Use this function as follows:

```
Kinvey.sharedClient.ping() { (result: Result<EnvironmentInfo, Swift.Error>) in
    switch result {
    case .success(let envInfo):
        print(envInfo)
    case .failure(let error):
        print(error)
    }
}
```

You should add this code to the `UIApplicationDelegate.application(_, didFinishLaunchingWithOptions:)` function after the Kinvey client is initialized. Then, run the app in Xcode.

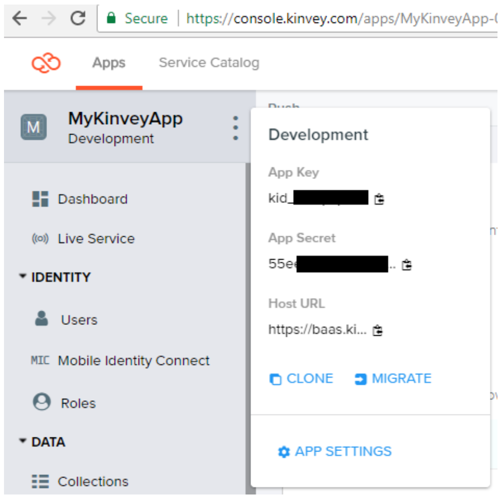
Add the Kinvey NativeScript client SDK

The Kinvey NativeScript client SDK is available in the NativeScript marketplace. To install, navigate to your NativeScript project directory in a command line window and run the following command:

```
tns plugin add kinvey-nativescript-sdk
```

Create a Kinvey client

1. Open the home page of your Kinvey app and click the ellipsis next to the app name. Then, copy the **App Key** and the **App Secret** values into a text file. You will need these values to initialize the Kinvey client.



2. Initialize the Kinvey client as follows:

```
import { Kinvey } from 'kinvey-nativescript-sdk';
  Kinvey.init({
      appKey: '<appKey>',
      appSecret: '<appSecret>'
  })
      }
 }
```

This is a TypeScript example. Click here to see the JavaScript example.

Test the connection with Kinvey

Before you develop your NativeScript app any further, you should test whether it is able to connect to your Kinvey app. The Kinvey client library enables you to test the connection through a `ping` method.

```
Kinvey.ping()
    .then((response) => {
        console.log(`Kinvey Ping Success. Kinvey Service is alive, version:
${response.version}, response: ${response.kinvey}`);
    })
    .catch((error) => {
        console.log(`Kinvey Ping Failed. Response: ${error.description}`);
    });
```

8

Step 5: Integrate your mobile app with the notification server

After you have set up your app to communicate with Kinvey, you need to integrate it with the notification server, so it can receive a device token for registration and also receive push notifications.

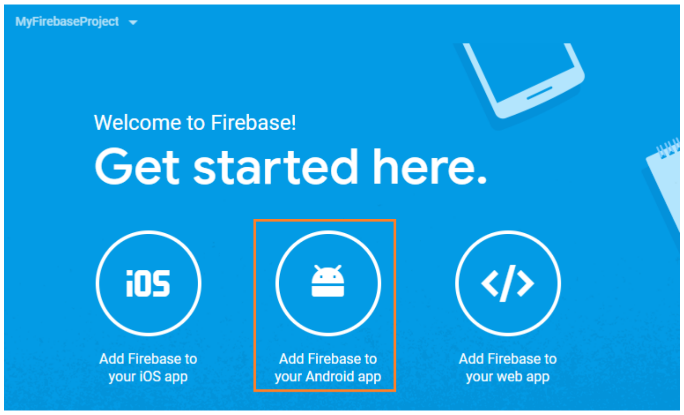
Integrate your app with GCM (for native Android and NativeScript Android apps)

You need to perform the following steps to integrate a native Android app developed in Android Studio with GCM. To do this, you will access the Firebase console.

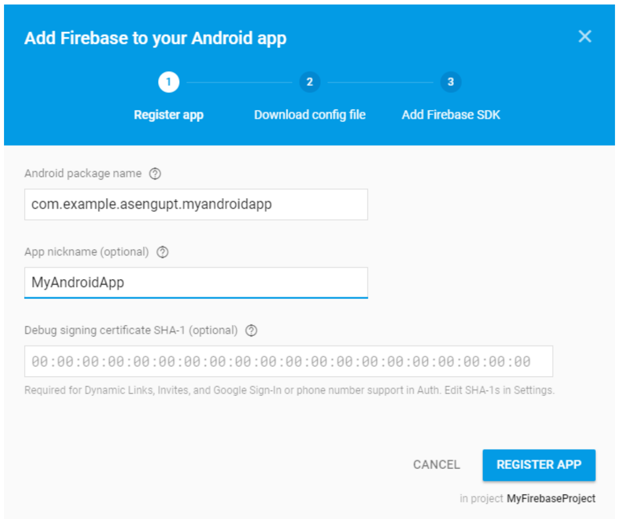
Note: For a NativeScript Android app, you need to perform only a subset of the steps in registering the app with Firebase.

Register the app with Firebase

1. In the Firebase project's home page click the **Add Firebase to your Android app** icon.


MyFirebaseProject
Welcome to Firebase!
Get started here.
Add Firebase to your iOS app
Add Firebase to your Android app
Add Firebase to your web app

2. Enter your Android or NativeScript package name and a name for the app in Firebase. For NativeScript, you will find the package name in the **package.json** file that is located in the **app** directory.



Add Firebase to your Android app

Android package name: com.example.asengupt.myandroidapp

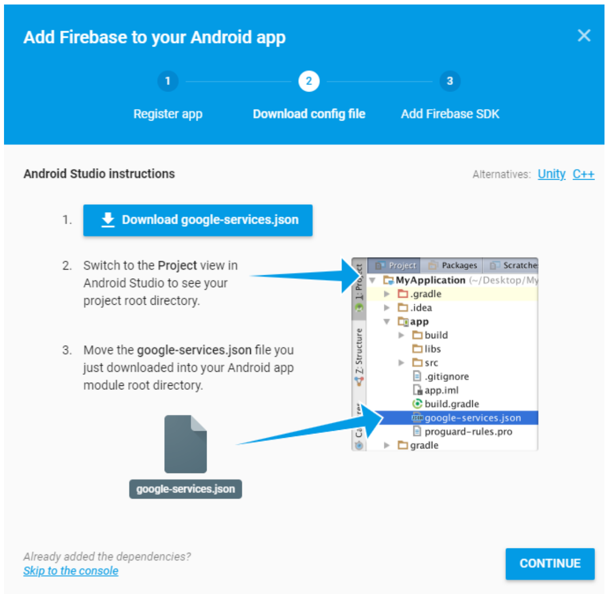
App nickname (optional): MyAndroidApp

Debug signing certificate SHA-1 (optional): 00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00

Required for Dynamic Links, Invites, and Google Sign-In or phone number support in Auth. Edit SHA-1s in Settings.

in project MyFirebaseProject

3. Follow the instructions to download the **google-services.json** file to the **app** directory in your Android project. If you are developing a NativeScript Android app, download and copy the JSON file to the `app\App_Resources\Android` directory in your NativeScript project.


Add Firebase to your Android app

1 Register app · 2 Download config file · 3 Add Firebase SDK

Android Studio instructions

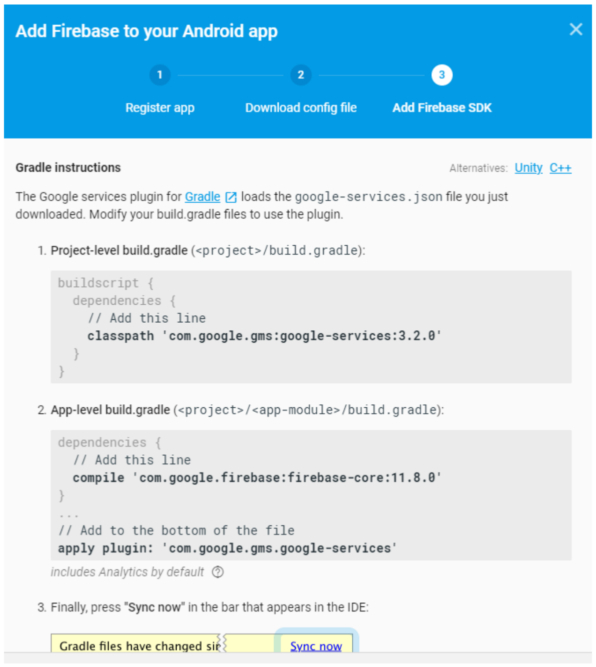
Alternatives: Unity C++

1. Download google-services.json
2. Switch to the **Project** view in Android Studio to see your project root directory.
3. Move the **google-services.json** file you just downloaded into your Android app module root directory.

Already added the dependencies?
Skip to the console

CONTINUE

4. Follow the instructions to add dependencies to your app and project-level **build.gradle** files and apply the Google services plugin. Skip this step if you are developing a NativeScript app.


Add Firebase to your Android app

1 Register app · 2 Download config file · 3 Add Firebase SDK

Gradle instructions

Alternatives: Unity C++

The Google services plugin for Gradle loads the `google-services.json` file you just downloaded. Modify your build.gradle files to use the plugin.

1. Project-level build.gradle (`<project>/build.gradle`):

```
buildscript {
  dependencies {
    // Add this line
    classpath 'com.google.gms:google-services:3.2.0'
  }
}
```

2. App-level build.gradle (`<project>/<app-module>/build.gradle`):

```
dependencies {
  // Add this line
  compile 'com.google.firebase:firebase-core:11.8.0'
}
...
// Add to the bottom of the file
apply plugin: 'com.google.gms.google-services'
```

includes Analytics by default

3. Finally, press "Sync now" in the bar that appears in the IDE:

Set GCM properties in the `kinvey.properties` file (native Android only)

Add the following lines to the `kinvey.properties` file (skip if you are developing a NativeScript app).

```
gcm.enabled=true
gcm.senderID=<Firebase_project_sender_ID>
```

Add the Google Play Services library (native Android only)

The Kinvey Client SDK internally uses the Google Play Services library. To add the library, perform the following steps (skip if you are developing a NativeScript app).

1. Add this line in the dependencies section of the app-level build.gradle file:

   ```
   implementation 'com.google.android.gms:play-services-gcm:11.8.0'
   ```

2. Add this line at the end of the same file:

   ```
   apply plugin: 'com.android.application'
   ```

3. When you see a prompt to sync the project, click **Sync Now**.

Integrate your app with Apple Push Notification Service (native iOS and NativeScript iOS only)

Enable push notifications for your app

Perform the following steps only if you are developing a native iOS app in Xcode.

1. In the project editor in Xcode, select your mobile app project and click **Capabilities**.
2. Click the **Push Notifications** switch to turn it **ON**.

If you are developing a NativeScript iOS app, open **platforms > ios > [AppName] > [AppName].entitlements** and add the following lines to the **dict** section:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>aps-environment</key>
    <string>development</string>
</dict>
</plist>
```

Register test devices

Push notifications do not get displayed on a Simulator. To test that your native iOS or NativeScript iOS app is receiving push notification messages you need to register an iPhone device and then create a provisioning profile.

1. Log in to the Apple developer portal.
2. Click **Certificates, Identifiers & Profiles**.
3. In the drop-down menu in the top left corner, verify that **iOS, tvOS, watchOS** is selected.
4. In the left-hand sidebar, select **Devices > All** and click **+**.
5. Provide a name for the device and the device UDID and click **Continue**.
6. Review the details for your device and click **Register**.

Create a provisioning profile

A provisioning profile enables you to install your app on a device. To create a development provisioning profile for your app, perform the following steps:

1. In the Apple developer portal, click **Certificates, Identifiers & Profiles** and verify that **iOS, tvOS, watchOS** is selected in the drop-down menu in the top left corner.
2. In the left-hand sidebar, select **Provisioning Profiles > Development** and click **+**.
3. Select **iOS App Development** and click **Continue**.
4. Select the **App ID** that you created earlier (to associate with the provisioning profile) and click **Continue**.
5. Select the Apple push notification certificate that you generated earlier and click **Continue**.
6. Select one or more devices to include in the provisioning profile and click **Continue**.
7. Provide a name for the profile and click **Continue**.
8. (Optional) Click **Download** to download the provisioning profile.
9. Click **Done**.

9

Step 6: Write code to handle push notifications and register users

The Kinvey client libraries for Android, iOS, and NativeScript provide the functionality to connect with the Kinvey backend app to register users and connect with the operating system to receive notifications from the notification server and display it in the notification pane.

For details, see the following topics:

- Handling push notifications in native Android apps
- Handling push notifications in native iOS apps
- Handling push notifications in NativeScript apps

Handling push notifications in native Android apps

The Kinvey client library provides a class named `KinveyGCMService` that provides event handlers to deal with push notifications. You must extend this class and bind it with Android's `WakefulBroadcastReceiver` class, that is responsible for listening to broadcast events such as push notifications.

Extend the KinveyGCMService class

Extend the `KinveyGCMService` class as shown in this example. At minimum, you need to implement the following methods:

- `public void onMessage(String message)`--this method is invoked when the app receives a push notification. You must implement this method to display the push notification.
- `public Class getReceiver()`--this method enables you to wire your app with the Android operating system. You must implement this method to return a class that extends Android's native `WakefulBroadcastReceiver` class.

```
import com.kinvey.android.push.KinveyGCMService;

public class GCMService extends KinveyGCMService {
    @Override
    public void onMessage(String message) {

        Log.i(TAG, "Push message received");
        displayNotification(message);
    }

    @Override
    //This method should return the WakefulBroadcastReceiver class you define in the
next step
    public Class getReceiver() {
        return GCMReceiver.class;
    }

    //Display the notification message using the NotificationCompat.Builder class

    private void displayNotification(String message){
        NotificationCompat.Builder mBuilder = new NotificationCompat.Builder(this)
                .setSmallIcon(R.drawable.ic_launcher_background)

.setContentTitle(getApplicationContext().getResources().getString(R.string.app_name))
                .setContentText(message);
        NotificationManager mNotificationManager = (NotificationManager)
getSystemService(Context.NOTIFICATION_SERVICE);
        mNotificationManager.notify(1, mBuilder.build());
    }
}
```

Extend the WakefulBroadcastReceiver class

The `WakefulBroadcastReceiver` class provides a `startWakefulService()` method that listens for push notifications.

```
import android.support.v4.content.WakefulBroadcastReceiver;

public class GCMReceiver extends WakefulBroadcastReceiver {
    public void onReceive(Context context, Intent intent) {
        ComponentName comp = new ComponentName(context.getPackageName(),
com.example.myandroidapp.GCMService.class.getName());
        startWakefulService(context, (intent.setComponent(comp)));
        setResultCode(Activity.RESULT_OK);
    }
}
```

Add permissions to your manifest

To use Google Cloud Messaging (GCM) for push notifications, you need to add the following permissions to the `<manifest>` tags in the `AndroidManifest.xml` file.

Note: Replace the `com.sample.myloginapp` package name with your own package name.

```
<uses-sdk android:minSdkVersion="15" android:targetSdkVersion="23"/>
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.WAKE_LOCK" />
<uses-permission android:name="com.google.android.c2dm.permission.RECEIVE" />
<permission android:name="com.sample.myloginapp.permission.C2D_MESSAGE"
    android:protectionLevel="signature" />
<uses-permission android:name="com.sample.myloginapp.C2D_MESSAGE" />
```

Next, within the `<application>` tag, register the **GCMReceiver** and **GCMService** that you created.

```
<receiver
        android:name="com.sample.myloginapp.GCMReceiver"
        android:permission="com.google.android.c2dm.permission.SEND" >
        <intent-filter>
            <action android:name="com.google.android.c2dm.intent.RECEIVE" />
            <category android:name="com.sample.myloginapp" />
        </intent-filter>
    </receiver>
    <service android:name=".GCMService" />
```

Register the user and device

The Kinvey client library provides a `push (Class)` convenience method to register an active user for push notifications. This method registers the user along with the device token, passing this information to Kinvey BaaS as well as the GCM server. You should call this method on the Kinvey client that you created earlier.

```
myKinveyClient.push(GCMService.class).initialize(getApplication());
```

Typically, you will want to call this method after the user has logged in successfully. For example:

```
UserStore.login(email, password, myKinveyClient, new KinveyClientCallback() {
   public void onSuccess(Object o) {
   myKinveyClient.push(GCMService.class).initialize(getApplication());
               }
```

Unregister the user and device

You may want to unregister the user for push notifications when the user logs out of the app. To do this, use the `disablePush()` method.

```
 logoutButton.setOnClickListener(new View.OnClickListener() {
            public void onClick(View view) {
  myKinveyClient.push(GCMService.class).disablePush();
              UserStore.logout(myKinveyClient, new KinveyClientCallback<Void>() {
```

Test push notifications using Kinvey

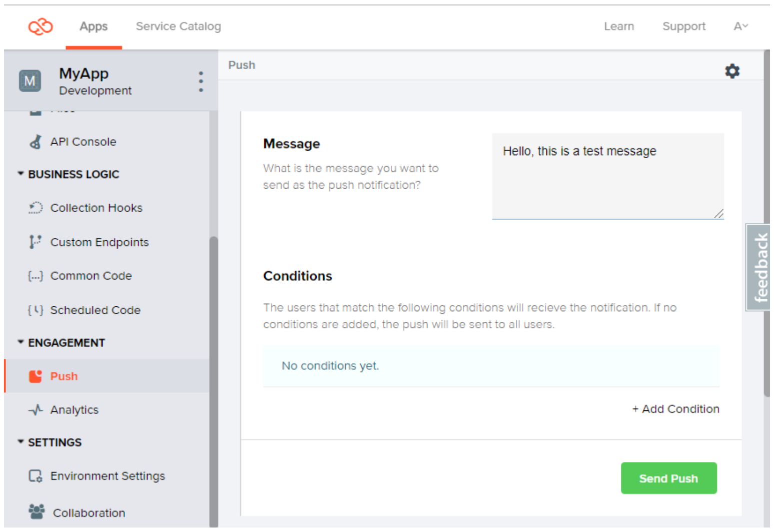
Your app should now be able to register active users and receive push notifications from GCM. To verify that your app is able to receive push notifications, create a few users and send test messages from your Kinvey BaaS app.

Follow these steps to create users:

1. In your Kinvey BaaS app, click **Users** under **IDENTITY**.
2. Click **+Add Users**.
3. In the **New App User** window, enter a username and password and click **Create**.

Then, run the Android app in an emulator and follow these steps to send push notifications:

1. Click **Push** under **ENGAGEMENT** in the left pane.
2. Compose a message and click **Send Push**.



Message

What is the message you want to send as the push notification?

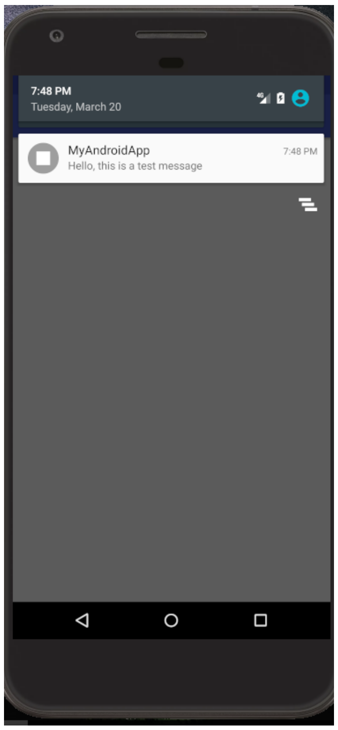
Hello, this is a test message

Conditions

The users that match the following conditions will recieve the notification. If no conditions are added, the push will be sent to all users.

No conditions yet.

You should be able to see the test message in your app.



7:48 PM
Tuesday, March 20

MyAndroidApp 7:48 PM
Hello, this is a test message

Handling push notifications in native iOS apps

You must use the Kinvey client that you initialized earlier to register users and display notifications.

Register users and devices

To register users and devices, use the `registerForPush()` method. This is usually done inside `application:didFinishLaunchingWithOptions:` after initializing the client.

```
if #available(iOS 10.0, *) {
       Kinvey.sharedClient.push.registerForNotifications(options: nil, completionHandler:
 completionHandler)
    } else {
        Kinvey.sharedClient.push.registerForPush(completionHandler: completionHandler)
    }
```

Display push notifications

To receive and handle push notifications, you must implement the `application(didReceiveRemoteNotification:)` method in your app delegate file,

```
func application(_ application: UIApplication, didReceiveRemoteNotification userInfo:
[AnyHashable : Any]) {
        let notification = JSON(userInfo["aps"])
        print("notification: \(String(describing: notification))")
        let topWindow = UIWindow(frame: UIScreen.main.bounds)
        topWindow.rootViewController = UIViewController()
        topWindow.windowLevel = UIWindowLevelAlert + 1
        let alert = UIAlertController(title: "Notification", message:
notification["alert"].stringValue, preferredStyle: UIAlertControllerStyle.alert)
        alert.addAction(UIAlertAction(title: "Done", style: UIAlertActionStyle.default,
 handler: {(_ action: UIAlertAction) -> Void in
            // continue your work
            // important to hide the window after work completed.
            // this also keeps a reference to the window until the action is invoked.
            topWindow.isHidden = true
        }))
        topWindow.makeKeyAndVisible()
        topWindow.rootViewController?.present(alert, animated: true, completion: {})
        }
```

Handling push notifications in NativeScript apps

The Kinvey client SDK for NativeScript provides easy-to-use methods to handle push notifications for Android as well as iOS devices.

Register the user and device

To register an active user and device (for both Android and iOS), you must use the `Push.register()` method on the Kinvey client that you initialized previously. This method registers the user along with the device token, passing this information to Kinvey BaaS app after retrieving the device token from Firebase or APNS. You will typically want to call this method after confirming that an active user has logged in.

```
const promise = Kinvey.Push.register({
  android: {
    senderID: '<Firebase project sender ID>'
  },
  ios: {
    alert: true,
    badge: true,
    sound: true
  }
})
  .then((deviceToken: string) => {
    // ...
  })
  .catch((error: Error) => {
    // ...
  });
```

Display the notification

To display the notification, use the `Push.onNotification()` method:

```
onNotification(): any {
       Kinvey.Push.onNotification((data: any) => {
           if ( typeof data === "string" ) {
               this.notificationData = JSON.parse(data);
           } else {
               this.notificationData = data;
           }
           dialogs.alert({
               title: this.notificationData.title,
               message: this.notificationData.message,
               okButtonText: "Close"
           }).then(() => {
               console.log("Dialog closed!");
           })
           .catch((error: Error) => {
               this.handleError(error.message, error);
           });
       });
  }
```

10

Step 7: Create a custom Kinvey BaaS endpoint

A custom endpoint in Kinvey BaaS is a JavaScript function that accepts an HTTP request from a client, processes it, and then returns an HTTP response. In the context of push notifications, you need to create and write a custom endpoint that accepts an HTTP request from an OpenEdge REST client and processes it by forming and sending a push notification message. You can write your endpoint to send push notification messages to both Android as well as iOS devices. The endpoint internally uses the push configurations defined in the **ENGAGEMENT > Push** section in the Kinvey app.

Create a custom endpoint

1. In your Kinvey BaaS app, click **Custom Endpoints** under **Business Logic**.
2. Click **+Add an Endpoint**.
3. In the dialog box that opens, enter a name for the endpoint, select **Code editor** and then click **Create an endpoint**.

When you create a custom endpoint, an `onRequest()` function is created by default.

```
function onRequest(request, response, modules) {
  response.complete();
}
```

The `onRequest()` function takes three parameters:

- `request`--a request object that the endpoint receives from a client. The request object contains fields like `method` (the HTTP method), `body`, `headers`, etc. The `body` field is populated by the custom properties received from the OpenEdge REST client. You can write your OpenEdge REST client to send a JSON request containing custom properties such as `id`, `message`, `from`, `subject`, etc. These properties can be accessed from in the endpoint's JavaScript function using the syntax `request.body.<propertyName>`. Typically, if you want to send a push notification to a specific user, your OpenEdge REST client will need to provide a user ID in a JSON message property that matches a user ID in Kinvey BaaS. You can then write endpoint code to send the push notification to only that user.
- `response`--a response object containing the HTTP status that the endpoint will send to the OpenEdge client.
- `modules`--Kinvey BaaS provides a large number of JavaScript libraries that you can use in your business logic. In our example, two of these libraries are important--`push`, which provides support for push notifications, and `collectionAccess`, which you can use to access users in Kinvey BaaS.

Initialize the `push` and `collectionAccess` libraries

Begin by initializing the `push` and `collectionAccess` libraries in the `onRequest()` function.

```
var push = modules.push, collectionAccess = modules.collectionAccess;
```

Form an Android message or a payload

A message is a single name-value pair (for example, `"message":"Hello World!"`). A payload is a JSON object. Here is an example of a payload:

```
var message = request.body.message;
   var from = request.body.from;
   var subject = request.body.subject;

  var androidPayload = {
         "message": message,
         "from": from,
         "subject": subject,
       };
```

Form an iOS message

An iOS message typically consists of the following properties:

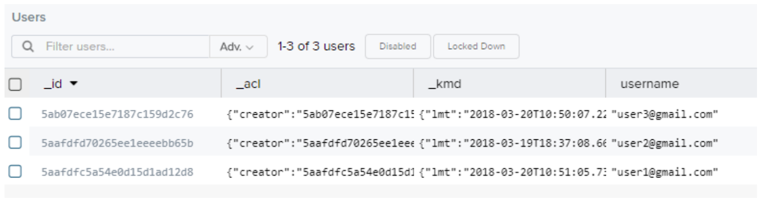
- alert--a JSON string or object containing the body of the push notification message
- badge--the value that is displayed in red circles on an app's icon on the device
- sound--the sound file that is to be played on the device when the notification is received.

```
var iOSAps = { alert: request.body.message, badge: request.body.count, sound: "default"
 };
    var iOSExtras = '';
```

Send or broadcast the message (or payload)

You can choose to either send a push notification to a specific user or group of users or broadcast it to all registered users and devices.

To send a message to a specific user or user group, you must write code in your OpenEdge REST client to pass the user ID. This user ID that is received from the OpenEdge client must match a user ID defined in Kinvey BaaS (identified by the `_id` column).



_id	_acl	_kmd	username
5ab07ece15e7187c159d2c76	{"creator":"5ab07ece15e7187c1	{"lmt":"2018-03-20T10:50:07.2	"user3@gmail.com"
5aafdfd70265ee1eeeebb65b	{"creator":"5aafdfd70265ee1ee	{"lmt":"2018-03-19T18:37:08.6	"user2@gmail.com"
5aafdfc5a54e0d15d1ad12d8	{"creator":"5aafdfc5a54e0d15d1	{"lmt":"2018-03-20T10:51:05.7	"user1@gmail.com"

Then, use the `collectionAccess` module to identify the user by matching the user ID with `request.body.id`.

```
collectionAccess.collection('user')
      .findOne({ "_id": collectionAccess.objectID(request.body.id) },
       function(err, user) {...});
```

Next, use the `push` module to send the message or payload.

```
push.sendPayload(user, iOSAps, iOSExtras, androidPayload, function(err, result) {...});
```

Note: You can use `null` for the `iOSAps` and `iOSExtras` parameters if you do not intend to send to iOS devices and vice versa.

Alternatively, if you want to send the push notification to all registered users and devices, use the `push` module to broadcast the message or payload.

```
push.broadcastPayload(iOSAps, iOSExtras, androidPayload, function(err, result) {...});
```

To learn more about writing custom endpoints, see the Kinvey Business Logic Reference documentation.

11

Step 8: Create an OpenEdge REST client

The OpenEdge REST client triggers a push notification by sending a message to the custom endpoint in Kinvey BaaS. Before you write the OpenEdge REST client, you should have the following data:

- The app secret and app key of the Kinvey BaaS app
- The name of the custom endpoint in Kinvey BaaS

In this example, we will use two OpenEdge procedure files--one to form an HTTP request and send it to the Kinvey BaaS app, and another to trigger the chain of events by calling the other procedure with a message.

Write an OpenEdge procedure to send the push notification to Kinvey BaaS

1. Create a procedure (.p) file in OpenEdge Developer Studio and import the following APIs.

   ```
   USING OpenEdge.Core.String.
   USING OpenEdge.Net.HTTP.*.
   USING OpenEdge.Net.URI.
   USING Progress.Json.ObjectModel.JsonObject.
   USING Progress.Lang.AppError.
   ```

   The `OpenEdge.Net.HTTP` API provides methods to create an HTTP client, send HTTP requests, and receive HTTP responses. The `OpenEdge.Net.URI` API enables you to form a URI that can be attached to the request. The `Progress.Json.ObjectModel.JsonObject` API provides methods to form the JSON messages that will be sent to Kinvey BaaS.

2. Create the required definitions.

   ```
   /*********************************************************************************
     DEFINITIONS
   ********************************************************************************/
   DEFINE INPUT PARAMETER lNotificatonBody AS LONGCHAR NO-UNDO.
   DEFINE VARIABLE oClient   AS IHttpClient   NO-UNDO.
   DEFINE VARIABLE oURI      AS URI           NO-UNDO.
   DEFINE VARIABLE oRequest  AS IHttpRequest  NO-UNDO.
   DEFINE VARIABLE oResponse AS IHttpResponse NO-UNDO.
   DEFINE VARIABLE vResp     AS CHARACTER     NO-UNDO.
   ```

3. Build the HTTP Client.

   ```
   oClient = ClientBuilder:Build():KeepCookies(CookieJarBuilder:Build():CookieJar):Client.
   ```

4. Build the URI. Replace <`your_kinvey_app_key`> with the app key of your Kinvey BaaS app and <`endpoint_name`> with the name of the custom endpoint in Kinvey.

   ```
   oURI = NEW URI('https','baas.kinvey.com').
   oURI:Path = '/rpc/<your_kinvey_app_key>/custom/<endpoint_name>'.
   ```

5. Build the request object. Replace <`kinvey_app_secret`> with the app secret of your Kinvey BaaS app.

   ```
   oRequest = RequestBuilder:Post(oURI,NEW String(lNotificatonBody))
        :AddHeader('Authorization','Basic <kinvey_app_secret>')
        :AcceptJson()
        :ContentType('application/json')
   /*     :WithData(NEW String(lNotificatonBody))*/
        :Request.
   ```

6. Build the response object.

   ```
   oResponse = ResponseBuilder:Build():Response.
   ```

7. Execute the HTTP client.

   ```
   oClient:Execute(oRequest,oResponse).
   ```

8. Handle errors that may result from the HTTP call.

   ```
   IF oResponse:StatusCode <> 200
       THEN
       vResp="Mobile notification Request Error:" + STRING(oResponse:StatusCode).
   ELSE
   DO:
       IF TYPE-OF(oResponse:Entity, JsonObject) THEN
   ```

```
            DEFINE VARIABLE resp AS LONGCHAR NO-UNDO.
        CAST(oResponse:Entity, JsonObject):Write(resp, TRUE).
        vResp=STRING(resp).
    END.
```

Write a procedure to trigger the push notification

Finally, create a procedure in OpenEdge Developer Studio to call the previous procedure with a message.

```
DEFINE VARIABLE temp AS CHARACTER NO-UNDO.
temp = '~{~"id":"5a8a6afcdace1e70f4f4f888","message":"Welcome to OE World"~}'.
RUN mobilenotification.p(INPUT temp).
```

In this example, the message contains a user ID that matches a user ID in Kinvey BaaS.

12

Push notification differences between Kinvey and Telerik Platform

The push notification mechanism in Kinvey addresses a broad range of requirements. However, there are a few differences between Kinvey and Telerik Platform that you may want to keep in mind when you are migrating your app from Telerik Platform to Kinvey:

- **Amazon SNS for iOS push notifications**—Kinvey uses Amazon SNS under the covers to communicate with Apple Push Notification Service.
- **Google Cloud Messaging (GCM) for Android push notifications**—Kinvey uses GCM to send Android push notifications.
- **Targeted notifications**—To send targeted notifications in Kinvey (for example by filtering for a group of users/devices), you must manually set system properties such as device model, OS version, etc.
- **Windows support**—Windows and Windows Phone are not supported in Kinvey.
- **Pushing messages from client**—In Kinvey, push notifications can only be sent through business logic defined in a custom endpoint or through the Kinvey Console (Web UI). Security policies are not present out of the box; you need to implement them in the business logic.
- **User-based registrations**—In Kinvey, registrations for push notifications are based on the user, not on the device. For this reason, Kinvey does not provide a UI-based browser for registered devices. However, you can view registered users in the Kinvey Console.
- **Scheduling push notifications**—Currently, Kinvey does not provide the option to schedule push notifications.
- **Debugging push notifications**—Kinvey does not provide debugging capabilities to identify failed push notifications.