Envió de notificaciones Push en Xamarin

**Table of Contents**

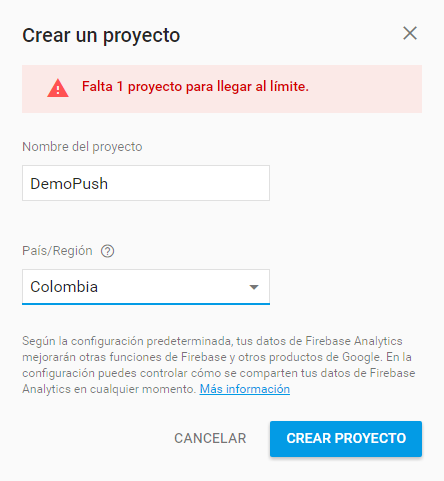
[**1**](#_gjdgxs) **Android 1**

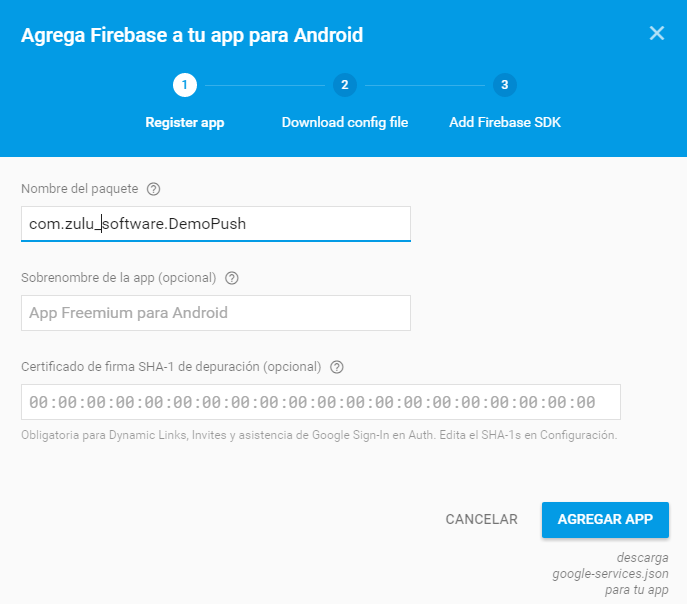
[**2**](#_30j0zll) **iOS 9**

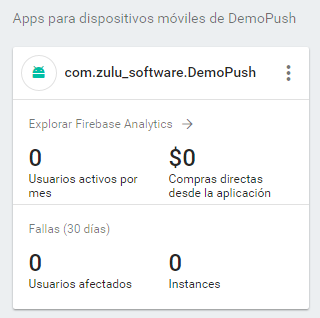
[**3**](#_3znysh7) **Test 27**

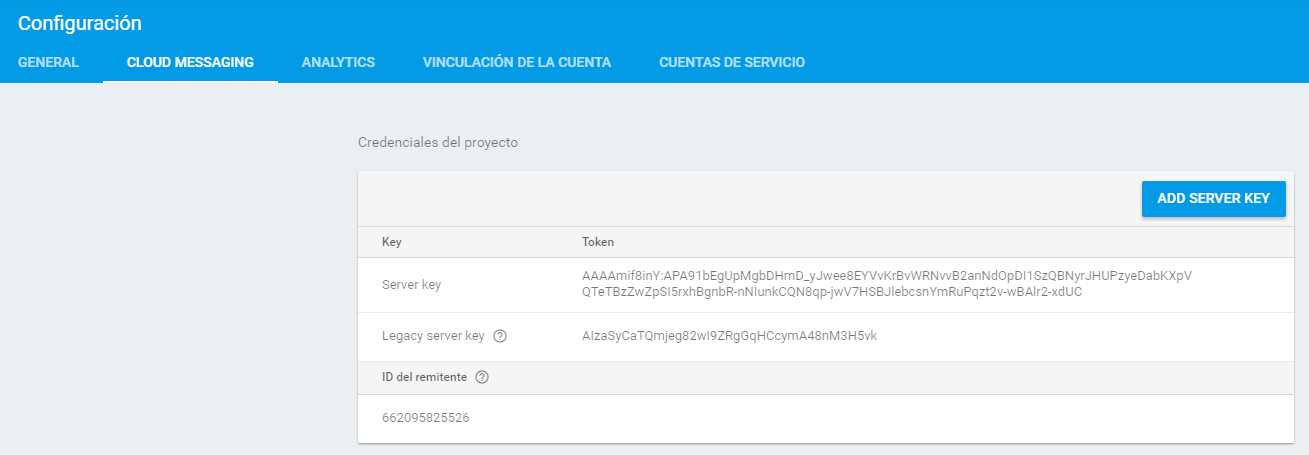
# Android

1. Configure un nuevo proyecto en <https://console.firebase.google.com/>. Es importante pasar a la pestaña de “Mensajería en la nube” y copiar la clave del servidor y el ID del remitente (los vamos a necesitar más adelante):





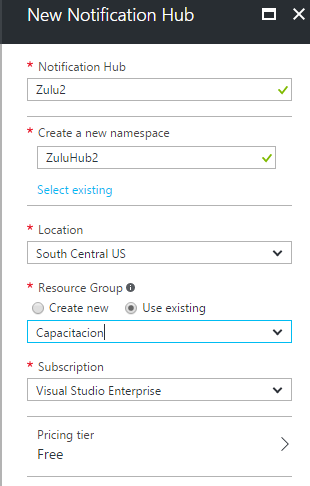


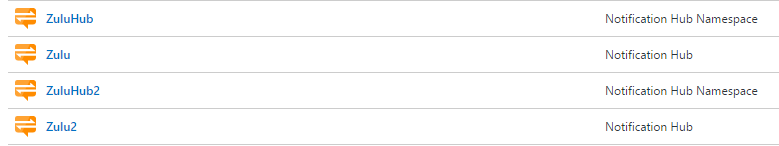


Servidor: AAAA788R8VI:APA91bEgD8wN-Dha9A0rSkJq7E7d3ecRL1iuAza44TVsen9BLuFRDI8j7KoRkPGpJIZZlwx1YMu7NX4RqxHWKhPEzcms7t0ZFOY2vzQ2-L\_sRjdW18qylELamNkgq95j6gsPvtkdCLHw

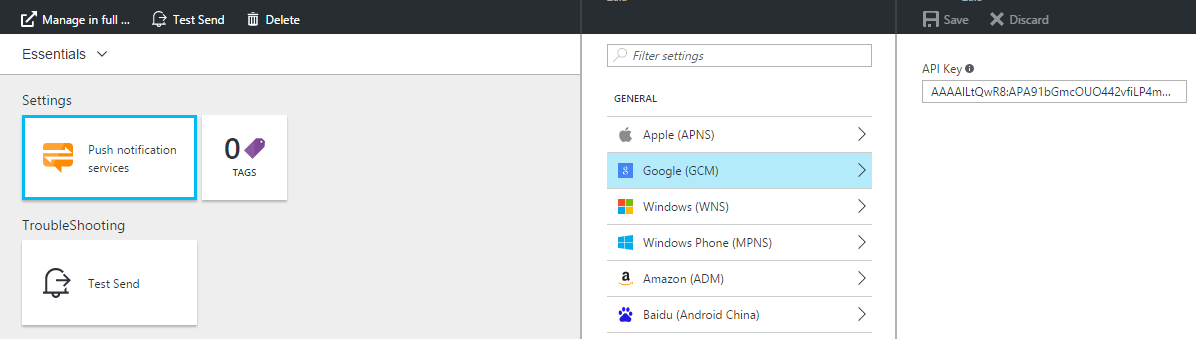
Remitente: 1029971243346

1. Configurar el Notification Hub en Azure. Click en **New**, luego: **Web + Mobile**, luego en: **Notification Hub**. Colocar un nombre al Hub y al Namespace. Debe quedar algo como esto:





Luego entrar al Hub y configurar la clave obtenida de Google en el primer paso:

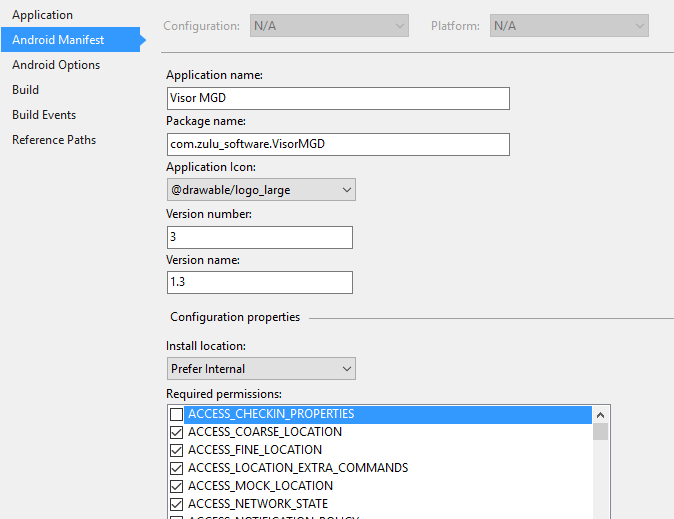


No cierres esto, porque vamos a necesitar copiar una cadena de conexión más luego

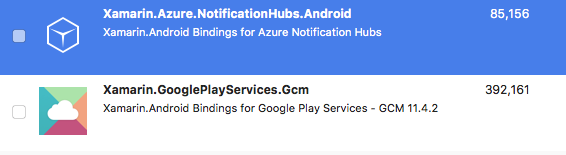
1. Crear la aplicación que se suscriba el Hub de notificaciones. Crea la aplicación Xamarin Forms, y asegúrate que esté funcionando el “Welcome to xamarin forms!”.

**Nota 1**: para estas pruebas vamos a necesitar un teléfono fijo, no sirve para emuladores.

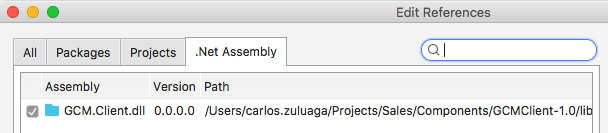
**Nota 2**: el proyecto debe tener un paquete, y el nombre de este debe empezar por una letra minúscula, esto es MUY importante. Por ejemplo:



1. En el proyecto Android, adicione los nuggets de **Xamarin.Azure.Notifications.Android** y **Xamarin.GooglePlayservices.Gcm**:



Adicional a esto, agregue la **GCM.Client.dll** que la puedes buscar en internet o sacar de mi repositorio:



1. En el proyecto Android, crea la clase **Constants** con el siguiente código:

namespace Sales.Droid

{

public static class Constants

{

public const string SenderID = "306406327100";

public const string ListenConnectionString = "Endpoint=sb://saleszuluhub.servicebus.windows.net/;SharedAccessKeyName=DefaultFullSharedAccessSignature;SharedAccessKey=R5/o5U0IuwNjYISkktWxK7mY8R4i9xQMtGtMlma1VOk=";

public const string NotificationHubName = "Sales";

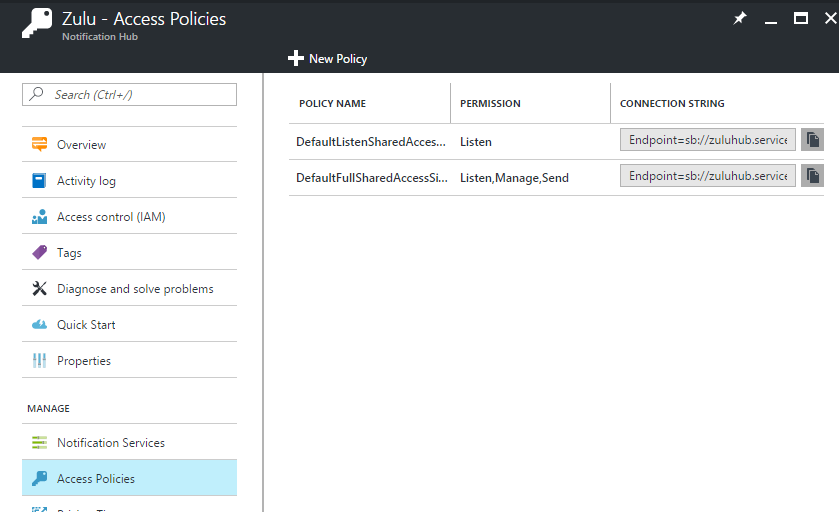
}

}

Reemplace el **SenderID** por el obtenido en el Firebase y el nombre del **Hub**:



Reemplace el **ListenConnectionString** por la configuración del Azure, del Hub en Access Polices:



Reemplace el **NotificationHubName** por el que se asignó en la configuración de Azure.

1. Modifique el **MainActivity.cs**:

using Android.App;

using Android.Content.PM;

using Android.OS;

namespace Soccer.Droid

{

[Activity(Label = "Soccer", Icon = "@drawable/ic\_launcher", Theme = "@style/MainTheme", MainLauncher = true, ConfigurationChanges = ConfigChanges.ScreenSize | ConfigChanges.Orientation)]

public class MainActivity : global::Xamarin.Forms.Platform.Android.FormsAppCompatActivity

{

#region Singleton

private static MainActivity instance;

public static MainActivity GetInstance()

{

if (instance == null)

{

instance = new MainActivity();

}

return instance;

}

#endregion

#region Methods

protected override void OnCreate(Bundle bundle)

{

instance = this;

TabLayoutResource = Resource.Layout.Tabbar;

ToolbarResource = Resource.Layout.Toolbar;

base.OnCreate(bundle);

global::Xamarin.Forms.Forms.Init(this, bundle);

LoadApplication(new App());

}

#endregion

}

}

1. En el proyecto compartido en el folder de **Interfaces**, crea la interfaz **IRegisterDevice**:

public interface IRegisterDevice

{

void RegisterDevice();

}

1. En el proyecto **Android** Adiciona la clase **RegistrationDevice**:

[assembly: Xamarin.Forms.Dependency(typeof(Sales.Droid.Implementations.RegistrationDevice))]

namespace Sales.Droid.Implementations

{

using Interfaces;

using Gcm.Client;

using Android.Util;

public class RegistrationDevice : IRegisterDevice

{

#region Methods

public void RegisterDevice()

{

var mainActivity = MainActivity.GetInstance();

GcmClient.CheckDevice(mainActivity);

GcmClient.CheckManifest(mainActivity);

Log.Info("MainActivity", "Registering...");

GcmClient.Register(mainActivity, Droid.Constants.SenderID);

}

#endregion

}

}

1. Crea la nueva clase: **MyBroadcastReceiver** con este código:

[assembly: Android.App.Permission(Name = "@PACKAGE\_NAME@.permission.C2D\_MESSAGE")]

[assembly: Android.App.UsesPermission(Name = "@PACKAGE\_NAME@.permission.C2D\_MESSAGE")]

[assembly: Android.App.UsesPermission(Name = "com.google.android.c2dm.permission.RECEIVE")]

[assembly: Android.App.UsesPermission(Name = "android.permission.GET\_ACCOUNTS")]

[assembly: Android.App.UsesPermission(Name = "android.permission.INTERNET")]

[assembly: Android.App.UsesPermission(Name = "android.permission.WAKE\_LOCK")]

namespace Sales.Droid

{

using System;

using System.Collections.Generic;

using System.Text;

using Android.App;

using Android.Content;

using Android.Util;

using Gcm.Client;

using ViewModels;

using WindowsAzure.Messaging;

[BroadcastReceiver(Permission = Gcm.Client.Constants.PERMISSION\_GCM\_INTENTS)]

[IntentFilter(new string[] { Gcm.Client.Constants.INTENT\_FROM\_GCM\_MESSAGE }, Categories = new string[] { "@PACKAGE\_NAME@" })]

[IntentFilter(new string[] { Gcm.Client.Constants.INTENT\_FROM\_GCM\_REGISTRATION\_CALLBACK }, Categories = new string[] { "@PACKAGE\_NAME@" })]

[IntentFilter(new string[] { Gcm.Client.Constants.INTENT\_FROM\_GCM\_LIBRARY\_RETRY }, Categories = new string[] { "@PACKAGE\_NAME@" })]

public class MyBroadcastReceiver : GcmBroadcastReceiverBase<PushHandlerService>

{

public static string[] SENDER\_IDS = new string[] { Constants.SenderID };

public const string TAG = "MyBroadcastReceiver-GCM";

}

[Service]

public class PushHandlerService : GcmServiceBase

{

#region Properties

public NotificationHub Hub { get; set; }

public static string RegistrationID { get; private set; }

#endregion

#region Methods

public PushHandlerService() : base(Constants.SenderID)

{

Log.Info(MyBroadcastReceiver.TAG, "PushHandlerService() constructor");

}

protected override void OnMessage(Context context, Intent intent)

{

Log.Info(MyBroadcastReceiver.TAG, "GCM Message Received!");

var msg = new StringBuilder();

if (intent != null && intent.Extras != null)

{

foreach (var key in intent.Extras.KeySet())

msg.AppendLine(key + "=" + intent.Extras.Get(key).ToString());

}

var message = intent.Extras.GetString("Message");

var type = intent.Extras.GetString("Type");

if (!string.IsNullOrEmpty(message))

{

var notification = intent.Extras.GetString("Notification");

createNotification("Sales App", message);

}

}

protected override bool OnRecoverableError(Context context, string errorId)

{

Log.Warn(MyBroadcastReceiver.TAG, "Recoverable Error: " + errorId);

return base.OnRecoverableError(context, errorId);

}

protected override void OnError(Context context, string errorId)

{

Log.Error(MyBroadcastReceiver.TAG, "GCM Error: " + errorId);

}

protected override void OnRegistered(Context context, string registrationId)

{

Log.Verbose(MyBroadcastReceiver.TAG, "GCM Registered: " + registrationId);

RegistrationID = registrationId;

Hub = new NotificationHub(Constants.NotificationHubName, Constants.ListenConnectionString, context);

try

{

Hub.UnregisterAll(registrationId);

}

catch (Exception ex)

{

Log.Error(MyBroadcastReceiver.TAG, ex.Message);

}

var tags = new List<string>() { };

var mainviewModel = MainViewModel.GetInstance();

if (mainviewModel.UserASP != null)

{

var userId = mainviewModel.UserASP.Id;

tags.Add("userId:" + userId);

}

try

{

var hubRegistration = Hub.Register(registrationId, tags.ToArray());

}

catch (Exception ex)

{

Log.Error(MyBroadcastReceiver.TAG, ex.Message);

}

}

protected override void OnUnRegistered(Context context, string registrationId)

{

Log.Verbose(MyBroadcastReceiver.TAG, "GCM Unregistered: " + registrationId);

createNotification("Sales App", "The device has been unregistered!");

}

void createNotification(string title, string desc)

{

//Create notification

var notificationManager = GetSystemService(Context.NotificationService) as NotificationManager;

//Create an intent to show UI

var uiIntent = new Intent(this, typeof(MainActivity));

//Create the notification

var notification = new Notification(Android.Resource.Drawable.SymActionEmail, title);

//Auto-cancel will remove the notification once the user touches it

notification.Flags = NotificationFlags.AutoCancel;

//Set the notification info

//we use the pending intent, passing our ui intent over, which will get called

//when the notification is tapped.

notification.SetLatestEventInfo(this, title, desc, PendingIntent.GetActivity(this, 0, uiIntent, 0));

//Show the notification

notificationManager.Notify(1, notification);

dialogNotify(title, desc);

}

protected void dialogNotify(String title, String message)

{

var mainActivity = MainActivity.GetInstance();

mainActivity.RunOnUiThread(() =>

{

AlertDialog.Builder dlg = new AlertDialog.Builder(mainActivity);

AlertDialog alert = dlg.Create();

alert.SetTitle(title);

alert.SetButton("Accept", delegate

{

alert.Dismiss();

});

alert.SetIcon(Resource.Drawable.ic\_launcher);

alert.SetMessage(message);

alert.Show();

});

}

#endregion

}

}

1. En la **MainViewModel** crea el método **RegisterDevice**:

public void RegisterDevice()

{

var register = DependencyService.Get<IRegisterDevice>();

register.RegisterDevice();

}

1. En la **LoginViewModel** llamar el método de registro, justo después que el usuario ingrese al sistema:

var userASP = (MyUserASP)response.Result;

MainViewModel.GetInstance().UserASP = userASP;

MainViewModel.GetInstance().RegisterDevice();

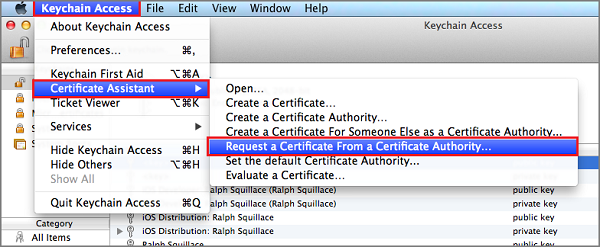
Settings.UserASP = JsonConvert.SerializeObject(userASP);

En el login por redes sociales y usuario recordado, también.

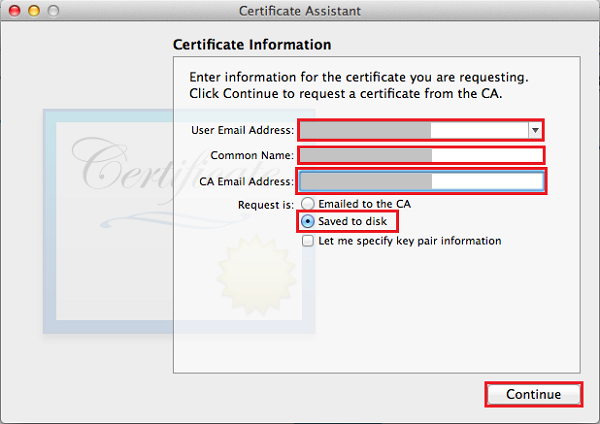
Por el momento con esto queda listo en Android, ahora vamos a configurar en iOS.

# iOS

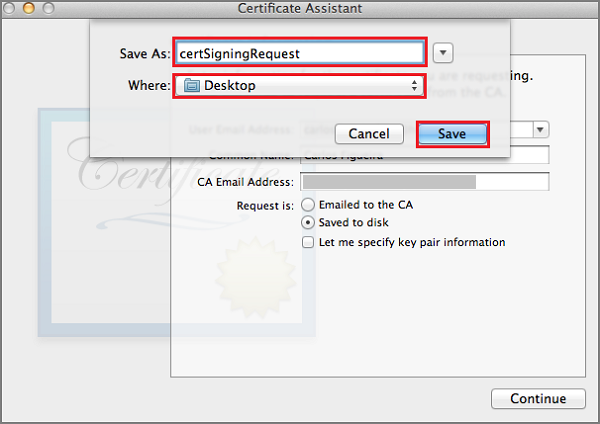
1. Click **Keychain Access**, expand **Certificate Assistant**, then click **Request a Certificate from a Certificate Authority...**.



Select your **User Email Address** and **Common Name** , make sure that **Saved to disk** is selected, and then click **Continue**. Leave the **CA Email Address** field blank as it is not required.



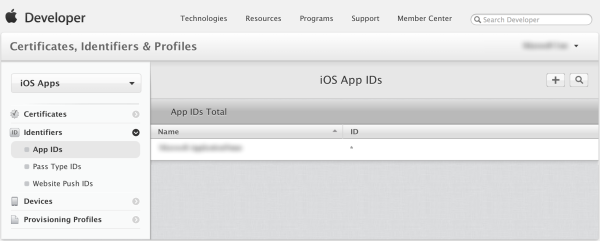
Type a name for the Certificate Signing Request (CSR) file in **Save As**, select the location in **Where**, then click **Save**.



This saves the CSR file in the selected location; the default location is in the Desktop. Remember the location chosen for this file.

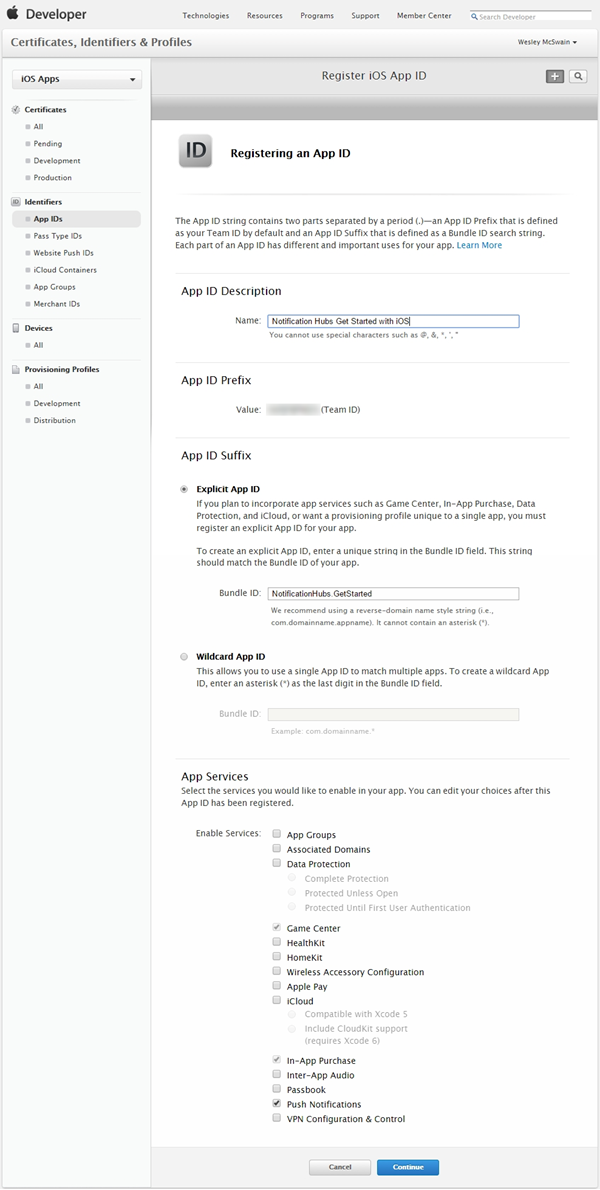
Next, you will register your app with Apple, enable push notifications, and upload this exported CSR to create a push certificate.

1. If you have not already registered your app, navigate to the [iOS Provisioning Portal](http://go.microsoft.com/fwlink/p/?LinkId=272456) at the Apple Developer Center, log on with your Apple ID, click **Identifiers**, then click **App IDs**, and finally click on the **+** sign to register a new app.

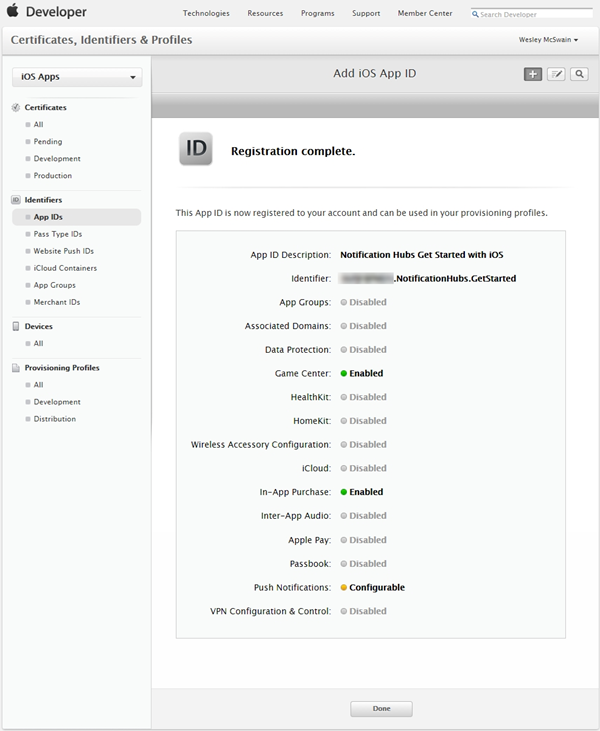


Update the following three fields for your new app and then click **Continue**:

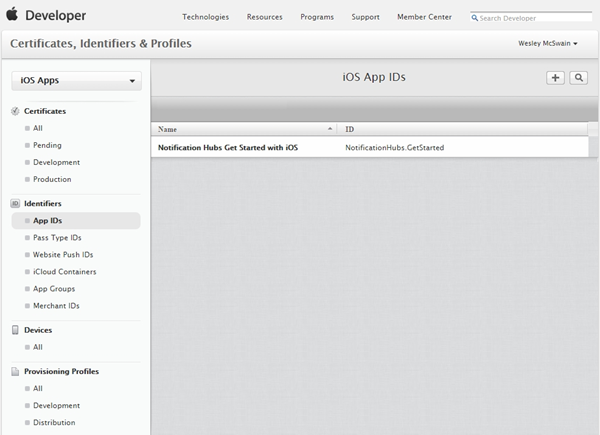
* **Name**: Type a descriptive name for your app in the **Name** field in the **App ID Description** section.
* **Bundle Identifier**: Under the **Explicit App ID** section, enter a **Bundle Identifier** in the form <Organization Identifier>.<Product Name>as mentioned in the [App Distribution Guide](https://developer.apple.com/library/mac/documentation/IDEs/Conceptual/AppDistributionGuide/ConfiguringYourApp/ConfiguringYourApp.html#//apple_ref/doc/uid/TP40012582-CH28-SW8). The *Organization Identifier*and *Product Name* you use must match the organization identifier and product name you will use when you create your XCode project. In the screeshot below *NotificationHubs* is used as a organization idenitifier and *GetStarted* is used as the product name. Making sure this matches the values you will use in your XCode project will allow you to use the correct publishing profile with XCode.
* **Push Notifications**: Check the **Push Notifications** option in the **App Services** section, .



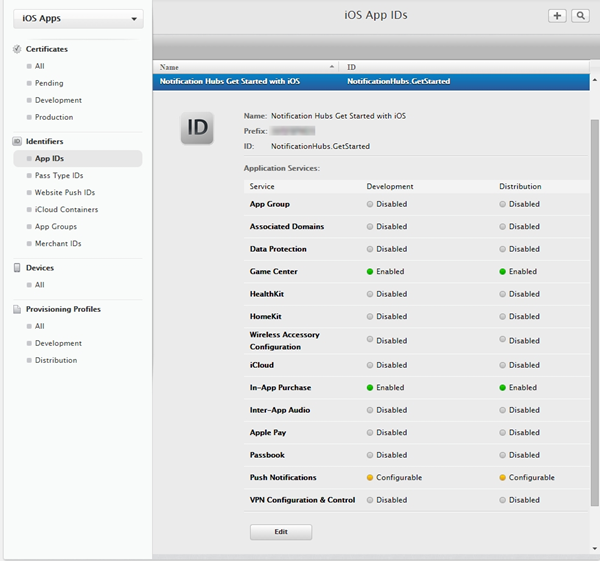
This generates your App ID and requests you to confirm the information. Click **Register** to confirm the new App ID. Once you click **Register**, you will see the **Registration complete** screen, as shown below. Click **Done**.



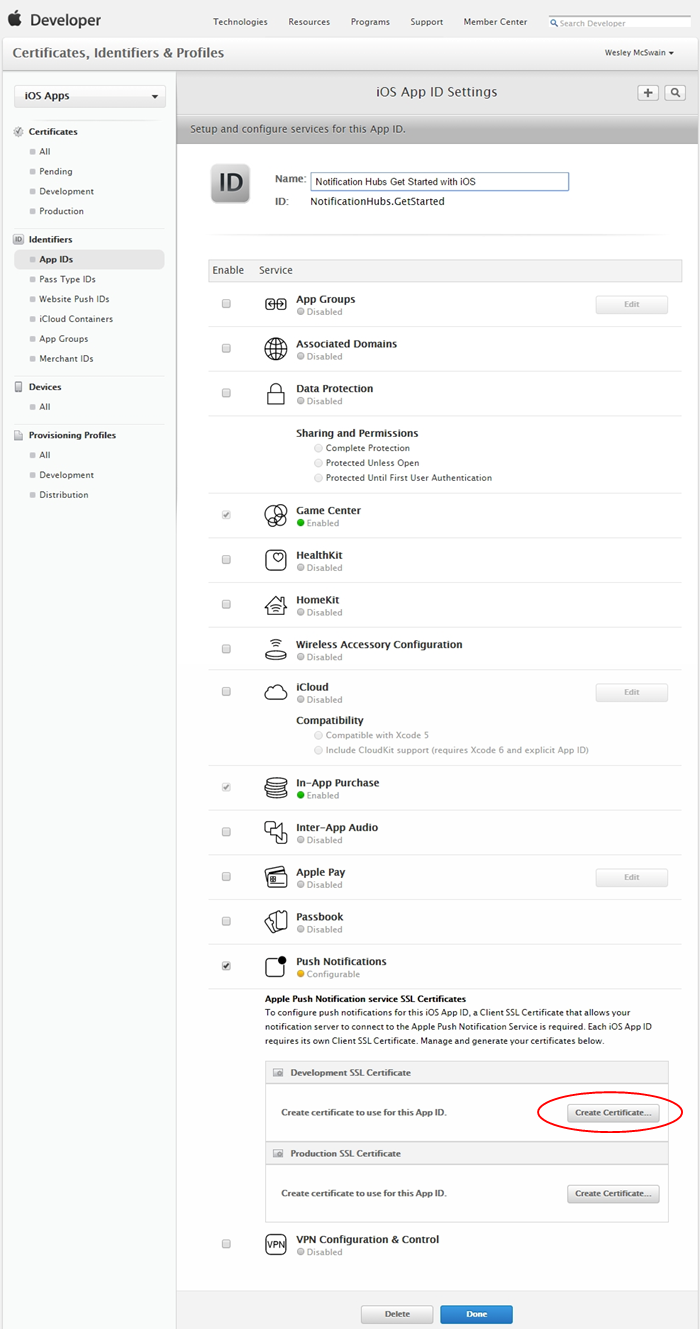
In the Developer Center, under App IDs, locate the app ID that you just created, and click on its row.



Clicking on the app ID will display the app details. Click the **Edit** button at the bottom.

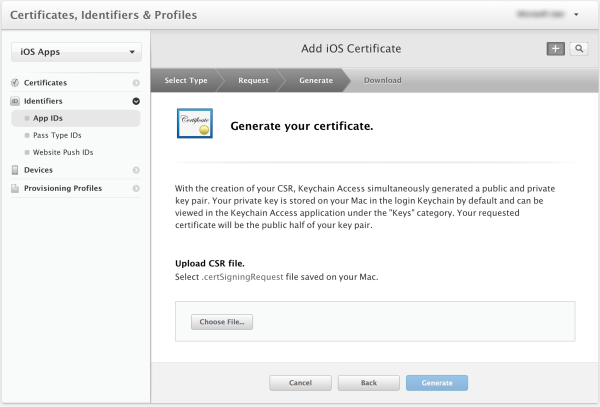


Scroll to the bottom of the screen, and click the **Create Certificate...** button under the section **Development Push SSL Certificate**.

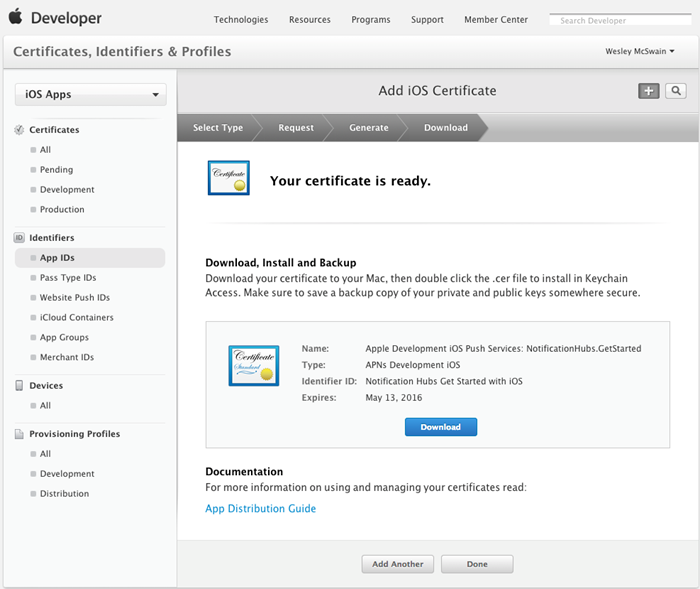


This displays the "Add iOS Certificate" assistant.

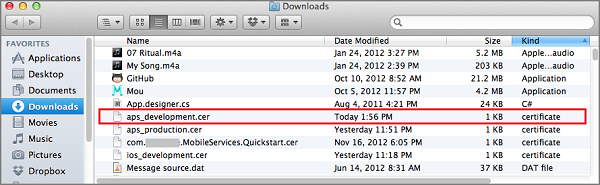
1. Click **Choose File**, browse to the location where you saved the CSR file that you created in the first task, then click **Generate**.



1. After the certificate is created by the portal, click the **Download** button, and click **Done**.

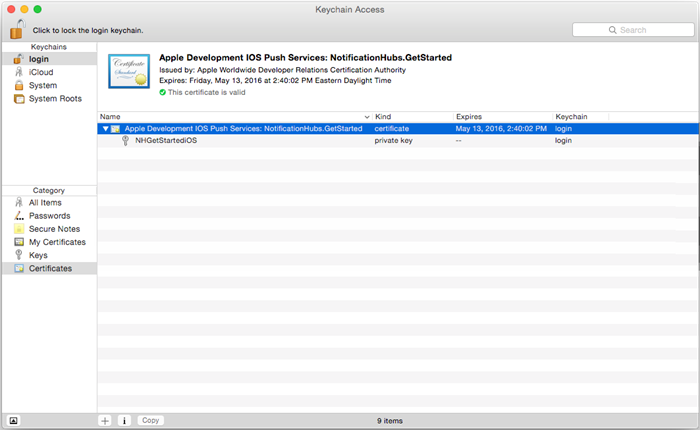


This downloads the certificate and saves it to your computer in your Downloads folder.

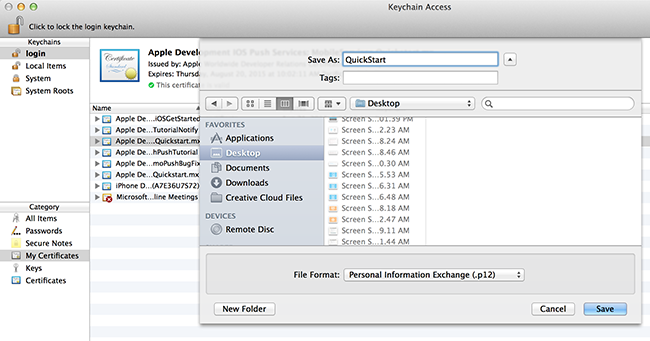


Double-click the downloaded push certificate **aps\_development.cer**.

This installs the new certificate in the Keychain, as shown below:

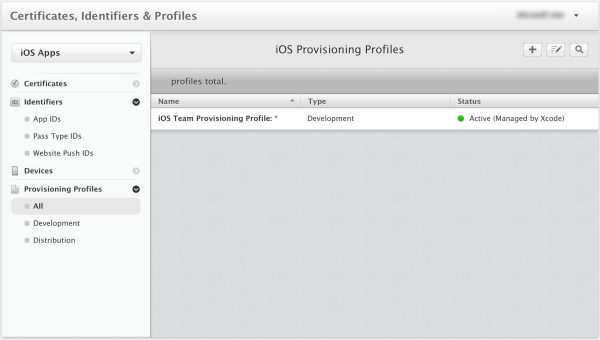


1. In Keychain Access, right-click the new push certificate that you created in the **Certificates** category. Click **Export**, name the file, select the **.p12** format, and then click **Save**.

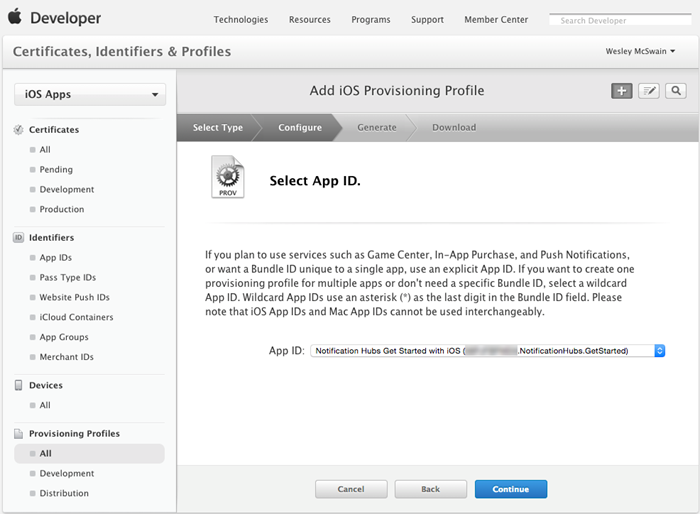


Make a note of the file name and location of the exported .p12 certificate. It will be used to enable authentication with APNS.

1. Back in the [iOS Provisioning Portal](http://go.microsoft.com/fwlink/p/?LinkId=272456), select **Provisioning Profiles**, select **All**, and then click the **+** button to create a new profile. This launches the **Add iOS Provisiong Profile** Wizard



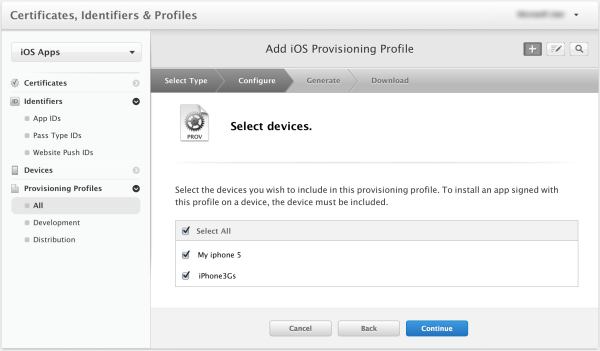
1. Select iOS App Development under Development as the provisiong profile type, and click Continue.
2. Next, select the app ID you just created from the App ID drop-down list, and click Continue



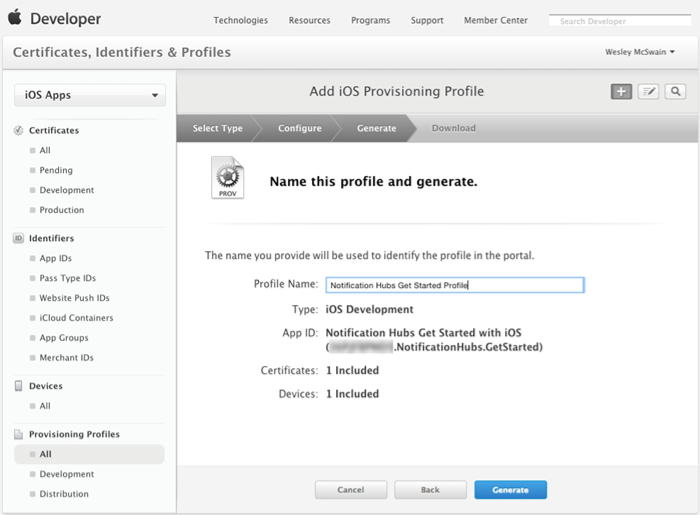
1. In the **Select certificates** screen, select your usual development certificate used for code signing, and click **Continue**. This is not the push certificate you just created.



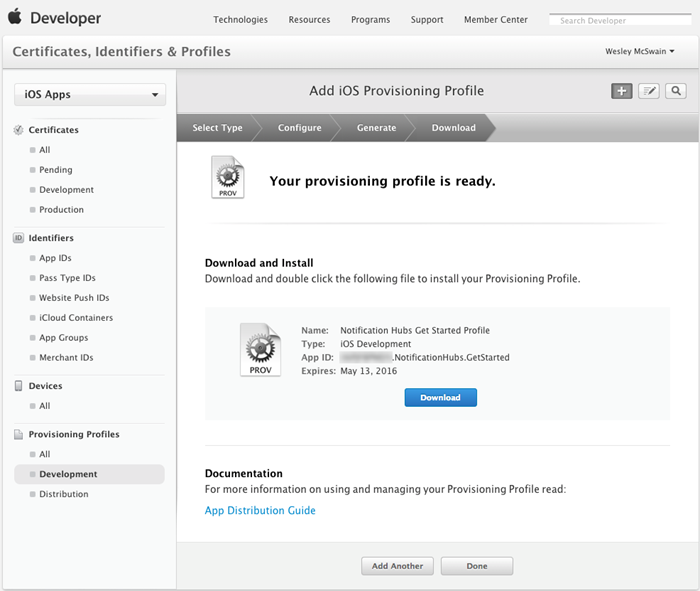
1. Next, select the **Devices** to use for testing, and click **Continue**



1. Finally, pick a name for the profile in Profile Name, click Generate.

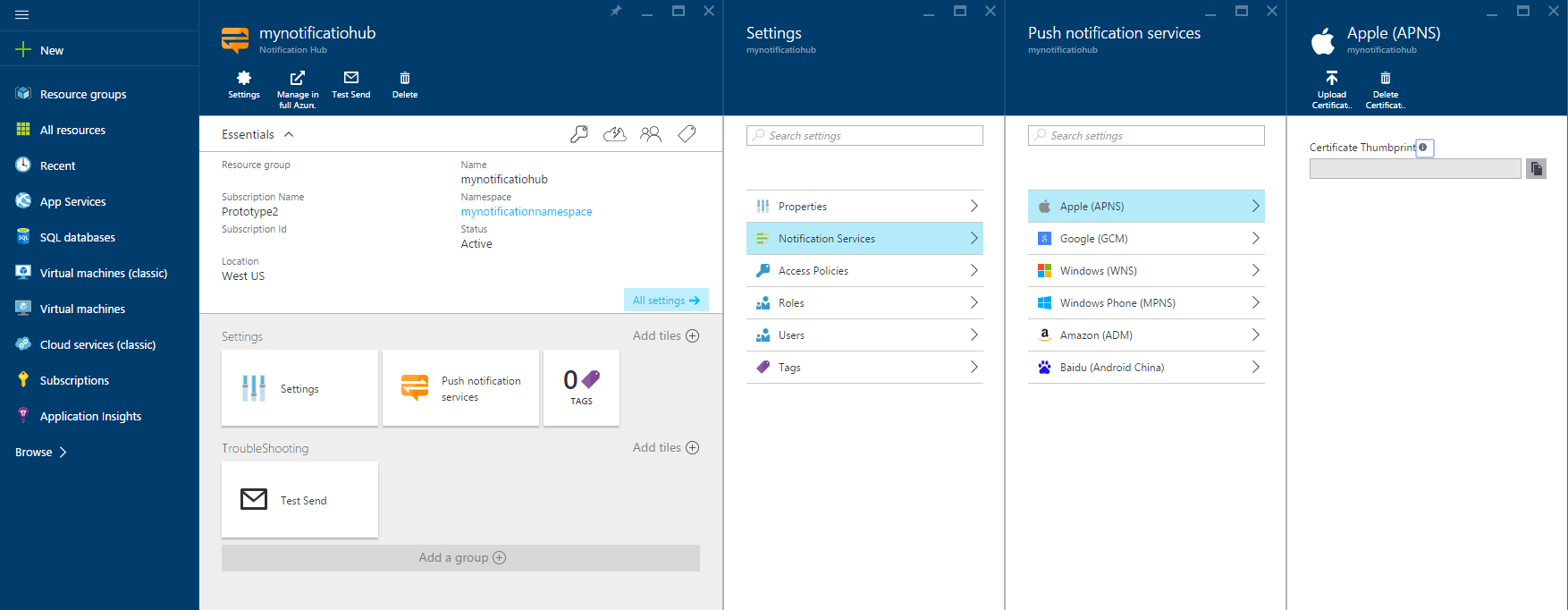


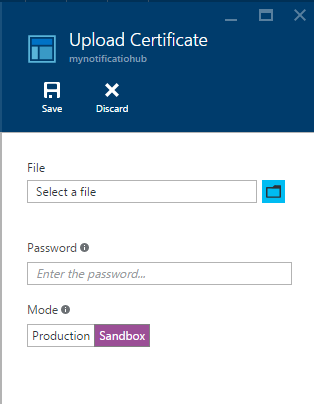
1. When the new provisioning profile is created click to download it and install it on your Xcode development machine. Then click **Done**.



1. As we want to configure the APNS connection, in the Azure Portal, open your Notification Hub settings, ande click on Notification Services, and then click the Apple (APNS) item in the list. Once done, click on Upload Certificate and select the .p12 certificate that you exported earlier, as well as the password for the certificate.

Make sure to select Sandbox mode since you will be sending push messages in a development environment. Only use the Production setting if you want to send push notifications to users who already purchased your app from the store.





Your notification hub is now configured to work with APNS, and you have the connection strings to register your app and send push notifications.

1. Crear la clase **Constans.cs**:

public class Constants  
{  
    // Azure app-specific connection string and hub path  
    public const string ConnectionString = "Endpoint=sb://tataapphub.servicebus.windows.net/;" +  
        "SharedAccessKeyName=DefaultFullSharedAccessSignature;" +  
        "SharedAccessKey=YU3JHje7mazlgYqlCvre8KbWIvL9vehKyYgM2JKya8U=";  
    public const string NotificationHubPath = "TataApp";  
}

1. Haga los siguientes cambios en el **AppDelegate.cs**:

namespace Sales.iOS

{

using System;

using System.Collections.Generic;

using Foundation;

using ImageCircle.Forms.Plugin.iOS;

using ViewModels;

using UIKit;

using WindowsAzure.Messaging;

[Register("AppDelegate")]

public partial class AppDelegate : global::Xamarin.Forms.Platform.iOS.FormsApplicationDelegate

{

public SBNotificationHub Hub { get; set; }

public override bool FinishedLaunching(UIApplication app, NSDictionary options)

{

global::Xamarin.Forms.Forms.Init();

ImageCircleRenderer.Init();

Xamarin.FormsMaps.Init();

LoadApplication(new App());

if (UIDevice.CurrentDevice.CheckSystemVersion(8, 0))

{

var pushSettings = UIUserNotificationSettings.GetSettingsForTypes(

UIUserNotificationType.Alert | UIUserNotificationType.Badge | UIUserNotificationType.Sound,

new NSSet());

UIApplication.SharedApplication.RegisterUserNotificationSettings(pushSettings);

UIApplication.SharedApplication.RegisterForRemoteNotifications();

}

else

{

UIRemoteNotificationType notificationTypes = UIRemoteNotificationType.Alert | UIRemoteNotificationType.Badge | UIRemoteNotificationType.Sound;

UIApplication.SharedApplication.RegisterForRemoteNotificationTypes(notificationTypes);

}

return base.FinishedLaunching(app, options);

}

public override void RegisteredForRemoteNotifications(UIApplication application, NSData deviceToken)

{

Hub = new SBNotificationHub(Constants.ConnectionString, Constants.NotificationHubPath);

Hub.UnregisterAllAsync(deviceToken, (error) =>

{

if (error != null)

{

Console.WriteLine("Error calling Unregister: {0}", error.ToString());

return;

}

var tags\_list = new List<string>() { };

var mainviewModel = MainViewModel.GetInstance();

if (mainviewModel.UserASP != null)

{

var userId = mainviewModel.UserASP.Id;

tags\_list.Add("userId:" + userId);

}

var tags = new NSSet(tags\_list.ToArray());

Hub.RegisterNativeAsync(deviceToken, tags, (errorCallback) =>

{

if (errorCallback != null)

Console.WriteLine("RegisterNativeAsync error: " + errorCallback.ToString());

});

});

}

public override void ReceivedRemoteNotification(UIApplication application, NSDictionary userInfo)

{

ProcessNotification(userInfo, false);

}

void ProcessNotification(NSDictionary options, bool fromFinishedLaunching)

{

if (null != options && options.ContainsKey(new NSString("aps")))

{

NSDictionary aps = options.ObjectForKey(new NSString("aps")) as NSDictionary;

string alert = string.Empty;

string type = string.Empty;

string notification = string.Empty;

if (aps.ContainsKey(new NSString("alert")))

{

alert = (aps[new NSString("alert")] as NSString).ToString();

}

//type = (aps[new NSString("Type")] as NSString).ToString();

if (!fromFinishedLaunching)

{

//notification = (aps[new NSString("Notification")] as NSString).ToString();

var avAlert = new UIAlertView("Sales App", alert, null, "Ok", null);

avAlert.Show();

}

}

}

public override void FailedToRegisterForRemoteNotifications(UIApplication application, NSError error)

{

new UIAlertView("Error registering push notifications", error.LocalizedDescription, null, "OK", null).Show();

}

}

}

1. Adicione este clase en el proyecto iOS:

[assembly: Xamarin.Forms.Dependency(typeof(Sales.iOS.Implementations.RegistrationDevice))]

namespace Sales.iOS.Implementations

{

using Foundation;

using Interfaces;

using UIKit;

public class RegistrationDevice : IRegisterDevice

{

#region Methods

public void RegisterDevice()

{

if (UIDevice.CurrentDevice.CheckSystemVersion(8, 0))

{

var pushSettings = UIUserNotificationSettings.GetSettingsForTypes(

UIUserNotificationType.Alert |

UIUserNotificationType.Badge |

UIUserNotificationType.Sound,

new NSSet());

UIApplication.SharedApplication.RegisterUserNotificationSettings(pushSettings);

UIApplication.SharedApplication.RegisterForRemoteNotifications();

}

else

{

UIRemoteNotificationType notificationTypes =

UIRemoteNotificationType.Alert |

UIRemoteNotificationType.Badge |

UIRemoteNotificationType.Sound;

UIApplication.SharedApplication.RegisterForRemoteNotificationTypes(notificationTypes);

}

}

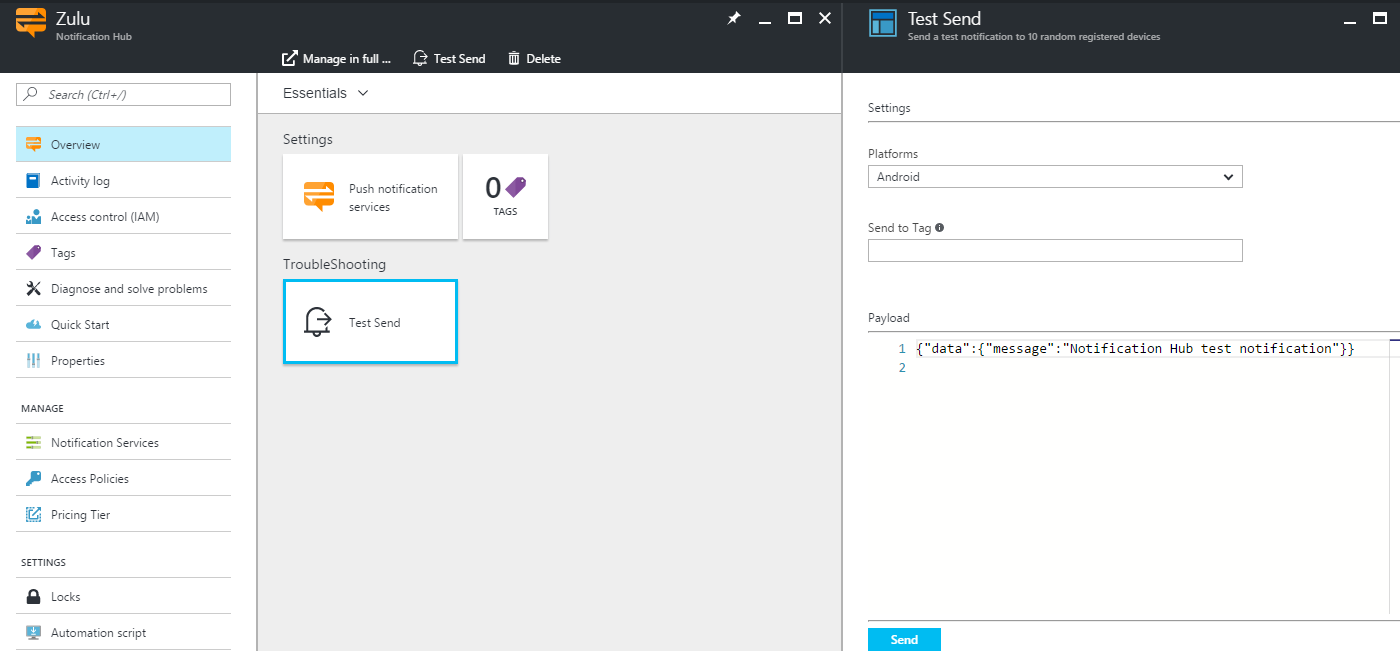
#endregion

}

}

# Test

1. Corra las aplicaciones (Android & iOS) en un dispositivo (no en emulador) e ingrese al Hub en el portal de Azure y envíe una notificación de prueba:



Verifique que el mensaje llegue con la aplicación corriendo, la aplicación minimizada y la aplicación cerrada.

1. Por último, creemos una aplicación de consola que envíe mensajes al Hub. Agregue este paquete a la aplicación de consola:

**Install-Package Microsoft.Azure.NotificationHubs**

1. Cambie el código de la clase Program por el siguiente:

using Microsoft.Azure.NotificationHubs;

using System;

namespace NotificationsSender

{

public class Program

{

private static NotificationHubClient hub;

public static void Main(string[] args)

{

hub = NotificationHubClient.CreateClientFromConnectionString("Endpoint=sb://zuluhub.servicebus.windows.net/;SharedAccessKeyName=RootManageSharedAccessKey;SharedAccessKey=SmhFDkgkbg5z5dKu2OTU7ds1iHw1i3BM5ifsHJ2rijk=", "Zulu");

do

{

Console.WriteLine("Type a new message:");

var message = Console.ReadLine();

SendNotificationAsync(message);

Console.WriteLine("The message was sent...");

} while (true);

}

private static async void SendNotificationAsync(string message)

{

await hub.SendGcmNativeNotificationAsync("{ \"data\" : {\"message\":\"" + message + "\"}}");

}

}

}

1. ¡Prueba tu aplicación!

Este documento es una adaptación del sitio: <https://docs.microsoft.com/en-us/azure/notification-hubs/xamarin-notification-hubs-push-notifications-android-gcm>