Azure Developer Immersions

When updates occur in the web site, you can raise push notifications to mobile and tablet apps. In this section, you will add code that uses an Azure Notification Hub to deliver messages.

> **Note** This walkthrough requires you to have a Microsoft account associated with an application development account because you need to be able to register apps with the store to receive push notifications; it is a similar situation with iOS. This is not free, unless you have an MSDN subscription. If you do not already have this and don't wish to pay the fee, which is currently $19, then you will not be able to get push notifications working and will skip this lab.

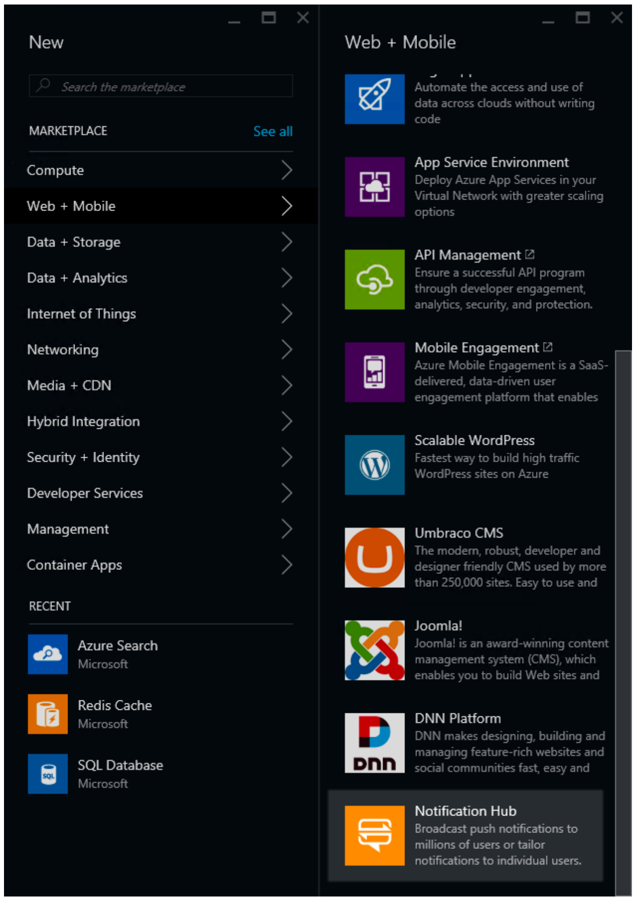
There is one exercise in this walkthrough:

1. Notification Hubs

Exercise 1: Notification Hubs

You will add Notification Hubs support in this exercise.

1. As before, in VSTS, move the correct Task to the In Progress state.
2. In the Azure portal, click **+ New**, select **Web + Mobile** and, in the list that appears, find and select **Notification Hub**.



Automate the access and use of data across clouds without writing code

App Service Environment
Deploy Azure App Services in your Virtual Network with greater scaling options

API Management
Ensure a successful API program through developer engagement, analytics, security, and protection.

Mobile Engagement
Azure Mobile Engagement is a SaaS-delivered, data-driven user engagement platform that enables

Scalable WordPress
Fastest way to build high traffic WordPress sites on Azure

Umbraco CMS
The modern, robust, developer and designer friendly CMS used by more than 250,000 sites. Easy to use and

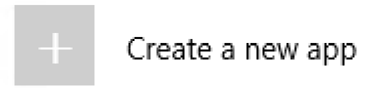
Joomla!
Joomla! is an award-winning content management system (CMS), which enables you to build Web sites and

DNN Platform
DNN makes designing, building and managing feature-rich websites and social communities fast, easy and

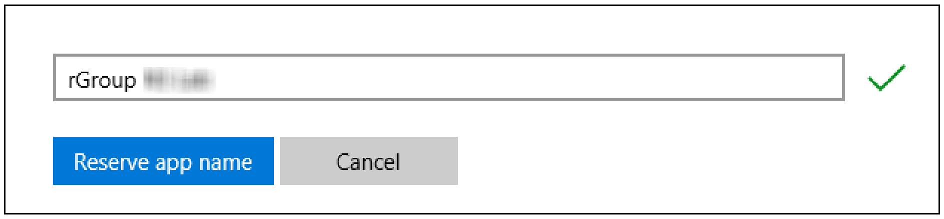
Notification Hub
Broadcast push notifications to millions of users or tailor notifications to individual users.

3. Call your hub **rGroup**.
4. Choose a namespace name, like **rgroupnsxxx** where xxx are your initials, bearing in mind it needs to be globally unique.
5. Change the region to match what you've been using.
6. Set the **Resource group** to **rGroup**.
7. Click **Create**. Azure will create the hub. This will take a little while. In the meantime, let's reserve a name for the mobile app; you need to do this before the app can receive notifications. In a separate browser window or tab, go to https://dev.windows.com/, log in with your Microsoft account, and click the **Dashboard** link.

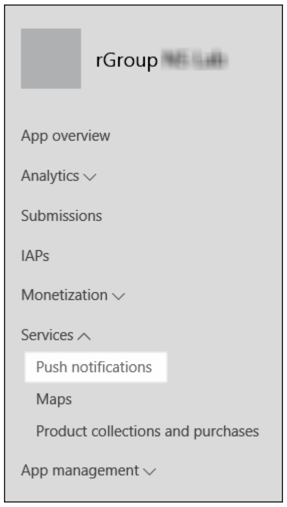
8. Click **Create a new app**.



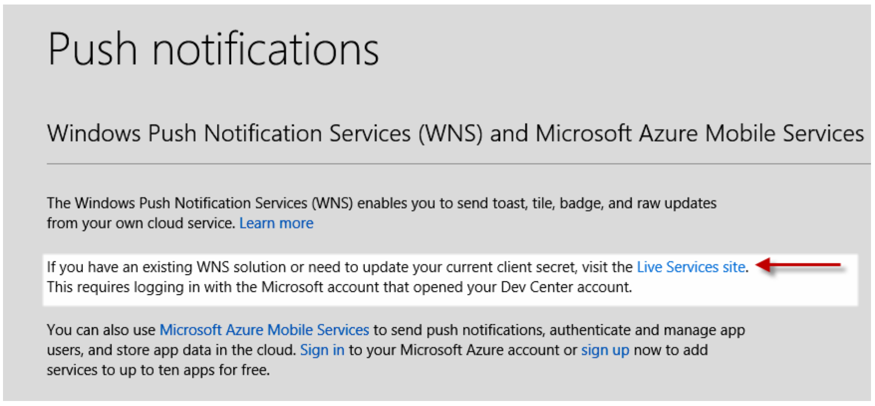
9. Type in a name for the app. This will need to be one that is not already taken. It will tell you if you chose one that is in use. Once you find one, click **Reserve app name**.



10. Once you are in the **App overview** screen, expand **Services** on the left and click **Push notifications**.



11. Click the link to the **Live Services site**. This takes you to a page for your new app. This is on the same site you used to set up Microsoft Account authentication earlier. However, you have to create the app through the Windows Dev Center to enable it for push notifications—apps created directly on the Microsoft Account Developer Center do not have that push support.



Push notifications

Windows Push Notification Services (WNS) and Microsoft Azure Mobile Services

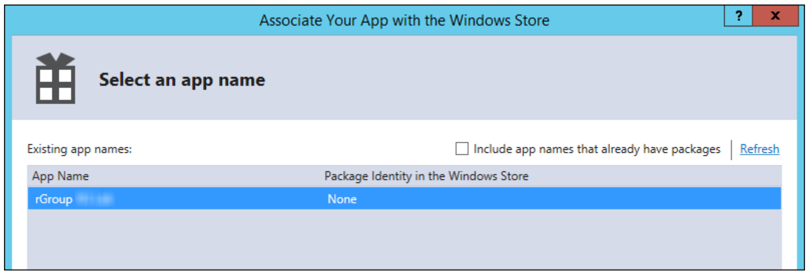
The Windows Push Notification Services (WNS) enables you to send toast, tile, badge, and raw updates from your own cloud service. Learn more

If you have an existing WNS solution or need to update your current client secret, visit the Live Services site. This requires logging in with the Microsoft account that opened your Dev Center account.

You can also use Microsoft Azure Mobile Services to send push notifications, authenticate and manage app users, and store app data in the cloud. Sign in to your Microsoft Azure account or sign up now to add services to up to ten apps for free.

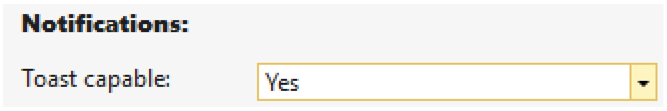
12. You should be on the **App Settings** page for your new app. Leave this page open and return to the browser window with the Azure portal—it should have finished creating your notification hub by now.
13. In the **rGroup** Notification hub's **Settings** pane, select **Notification Services**. In the **Push notification services** pane that opens, select **Windows (WNS)**.

14. Copy the **Package SID** from the **App Settings** page on the Microsoft Account Developer Center to the **Package SID** field in the **Windows (WNS)** pane in the Azure portal.
15. Copy the **Client secret** from the Microsoft Account Developer Center to the **Security Key** field in the Azure portal.
16. Click **Save** in the **Windows (WNS)** tab.
17. Next, you need to associate your client app with the application you just created in the dev center so that it is able to receive notifications delivered to that app. In Visual Studio, right-click on the **Rg.ClientApp** project in **Solution Explorer**, and select **Store | Associate App with the Store**.
18. In the Wizard click **Next**.
19. Sign in and select the app you just created from the list.


Associate Your App with the Windows Store
?
x
Select an app name
Existing app names:
Include app names that already have packages
Refresh

App Name	Package Identity in the Windows Store
rGroup	None

20. Click **Next**, then click **Associate**.
21. Double-click the project's **Package.appxmanifest** file, then select **Yes** under **Toast capable**.



Notifications:

Toast capable: Yes

22. In addition to associating the app with the store and enabling toast popups, you also need the app to register for notifications at runtime; the Windows Notification Service needs to know where the app is running to be able to deliver notifications. Open **App.xaml.cs** and add the following using directive:

```
using Windows.Networking.PushNotifications;
```

23. Add these members to the **App** class:

```
public static PushNotificationChannel PushChannel { get; private set; }

private async void InitNotificationsAsync()
{
    PushChannel = await PushNotificationChannelManager
        .CreatePushNotificationChannelForApplicationAsync();

}
```

24. Find the **OnLaunched** method and add this as the first line:

```
InitNotificationsAsync();
```

That means your app will now be able to receive push notifications and show toast if they arrive while it is not running, but you are not quite done. Your Azure Notification Hub needs to know about this device registration. There is already an endpoint for this but it is not fully implemented. Before you fill that in, make the client supplu the details of its push notification registration to that endpoint

25. Double-click **MainPage.xaml**, add a button with text of **Register for Notifications**, and set its **x:Name** to **registerForNotificationsButton**. You can use the following XAML.

```
<Button x:Name="registerForNotificationsButton" Content="Register for Notifications"
HorizontalAlignment="Left" Margin="50,150,0,0" VerticalAlignment="Top"/>
```

26. Add a click handler.
27. Add the following using statement:

```
using Windows.UI.Popups;
```

28. Add the **async** keyword before the handler's **void** keyword.
29. Add the following code:

```
var req = new NotificationRegistration
{
    Platform = "wns",
    Handle = App.PushChannel.Uri
};
await _apiClient.Notification.PostByRegistrationAsync(req);
var md = new MessageDialog("Done", "Registration");
await md.ShowAsync();
```

30. Now you can implement the back end. In the **Rg.Api** project's **NotificationController.cs** file, you will find the registration endpoint used by the code you just added to the client. As you can see, it calls **NotificationOperations.RegisterAsync**.

31. You can find that in the **Rg.ServiceCore** project's **Operation** folder in the **NotificationOperations.cs** file. Open that.
32. Right-click on the **Rg.ServiceCore** project's **References** node and select **Manage NuGet Packages**.
33. Search for the **Microsoft.Azure.NotificationHubs** package and install it.
34. At the top of **NotificationOperations.cs**, add these using directives:

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Azure.NotificationHubs;
```

35. You need to define some message "templates". These define the basic structure of the notifications you will be sending, with placeholders for the data that is different for each message. This enables us to handle multi-platform messaging. You define suitable templates for each platform and, when it comes to raising notifications, you just provide the values to plug into placeholders. The Azure Notification Hub will work out which device types are currently registered for notifications and will work out which templates it needs to use when it comes to sending the messages. Add the following string constants inside the **NotificationOperations** class:

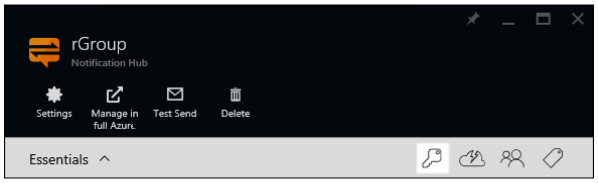
```
private const string MpnsToastTemplate = "<?xml version=\"1.0\" encoding=\"utf-8\"?>"
+
                "<wp:Notification xmlns:wp=\"WPNotification\">" +
                    "<wp:Toast>" +
                        "<wp:Text1>$(message)</wp:Text1>" +
                    "</wp:Toast> " +
                "</wp:Notification>";

private const string WnsToastTemplate = @"<toast><visual><binding
template=""ToastText01""><text
id=""1"">$(message)</text></binding></visual></toast>";
private const string ApnsToastTemplate = "{\"aps\":{\"alert\":\"$(message)\"}}";
private const string GcmToastTemplate = "{\"data\":{\"msg\":\"$(message)\"}}";
```

This covers both of Microsoft's systems, iOS, and Android. Each of these templates has a single placeholder: **message**.

36. Add a field to hold a reference to the notification hub client like this:

```
private static readonly NotificationHubClient _hub =
NotificationHubClient.CreateClientFromConnectionString(
    "CONNECTION STRING",
    "HUB NAME");
```

37. Of course, these are not the real connection string and hub. For the name, fill in the value you entered when you created the hub. You can get the connection string by clicking on the key icon in your notification hub's pane in the Azure portal.



This will show two connection strings.
You want the **DefaultFullSharedAccessSignature** one with **Listen**, **Manage**, and **Send** permission.

38. Find the **RegisterAsync** method and add the **async** keyword after the **static** keyword.

39. Replace the body of the method with this code:

```
Trace.TraceInformation("RegisterAsync");
Trace.TraceInformation($"RegisterAsync: {user.UserInfoId} ({user.Name}),
{registration.Platform}: {registration.Handle}");

var existingRegistrations = (await
_hub.GetRegistrationsByChannelAsync(registration.Handle, 100))
    .ToList();
Trace.TraceInformation($"Found {existingRegistrations.Count} registrations");
if (existingRegistrations.Count > 1)
{
    foreach (RegistrationDescription existingRegistration in
existingRegistrations.Skip(1))
    {
        Trace.TraceInformation($"Deleting {existingRegistration.RegistrationId}");
        await _hub.DeleteRegistrationAsync(existingRegistration);
    }
}

string registrationId;
if (existingRegistrations.Count != 0)
{
    registrationId = existingRegistrations[0].RegistrationId;
    Trace.TraceInformation($"Using existing registration {registrationId}");
}
else
{
    Trace.TraceInformation("Creating new registration id...");
    registrationId = await _hub.CreateRegistrationIdAsync();
    Trace.TraceInformation($"New registration: {registrationId}");
}

RegistrationDescription registrationDescription;
Trace.TraceInformation($"Registering template for {registration.Platform}");
switch (registration.Platform)
{
    case "mpns":
        registrationDescription = new
MpnsTemplateRegistrationDescription(registration.Handle, MpnsToastTemplate);
```

```
        break;
    case "wns":
        registrationDescription = new
WindowsTemplateRegistrationDescription(registration.Handle, WnsToastTemplate);
        break;
    case "apns":
        registrationDescription = new
AppleTemplateRegistrationDescription(registration.Handle, ApnsToastTemplate);
        break;
    case "gcm":
        registrationDescription = new
GcmTemplateRegistrationDescription(registration.Handle, GcmToastTemplate);
        break;
    default:
        return false;
}

registrationDescription.RegistrationId = registrationId;
registrationDescription.Tags = new HashSet<string> { "userId:" + user.UserInfoId };
try
{
    Trace.TraceInformation("Creating or updating registration");
    await _hub.CreateOrUpdateRegistrationAsync(registrationDescription);
    Trace.TraceInformation("Created or updated registration");
}
catch (Exception e)
{
    // Should we handle MessagingException separately?
    Trace.TraceInformation("Error creating or updating registration: " + e);
    throw;
}

return true;
```

This code starts by cleaning up any duplicate registrations that may already exist, leaving just one for the device in question. If there are none, it creates one. It then picks a template type suitable for the type of device making the registration and tells the Notification Hub to associate that template with this particular device registration. It also sets a tag containing the user id; this enables us to target a particular user. Without tags, notifications are broadcast to all devices registered to the hub. This way, you can deliver messages to all of the devices that a particular individual is using.

40. Find the **NotifyAsync** method. This is where you will send out notifications.
41. Add the **async** keyword after the **static** keyword.

42. Replace the body of the method with this code:

```
await _hub.SendTemplateNotificationAsync(
    new Dictionary<string, string> { { "message", message } },
    "userId:" + user.UserInfoId);
```

This tells the Notification Hub to deliver a notification to all devices registered with the specified user ID tag. The dictionary supplies all the values to put in the placeholders in the relevant templates. In this case, just a single value: the message text. The hub will find all of the registrations with that tag and, for each one, will know to use the correct template type for the relevant device because you told it which kind of device is being registered in the **RegisterAsync** method.

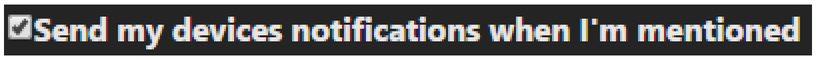
43. Finally, you need to arrange to call this method. Open the **UserOperations.cs** file in the **Rg.ServiceCore** project.
44. Find the **NotifyMentionsAsync** method, which you modified in an earlier section.
45. In this method you'll add a **foreach** loop that iterates over all the users mentioned in a message or comment. You'll add code that notifies subscribed users.

```
IList<PushTriggerRegistration> mentionTriggerRegistrations = await
dbContext.PushTriggerRegistrations
    .Where(r => r.TypeId == TriggerType.Mention)
    .ToListAsync();

     string mentioningUserName = await dbContext.UserInfos
            .Where(u => u.UserInfoId == mentioningUserId)
            .Select(u => u.Name)
            .SingleAsync();

foreach (UserInfo mentionedUser in text.MentionsUser)
{
  if
(mentionedUser.MentionNotificationSettings.HasFlag(NotifyOptions.PushNotification))
  {
     await NotificationOperations.NotifyAsync(
           mentionedUser,
           $"{mentioningUserName} mentioned you in a {mentionedIn}");
  }
}
```

This checks for another user setting. If the user has opted in to receiving push notifications, this calls the **NotifyAsync** method you just implemented to send out push notifications.

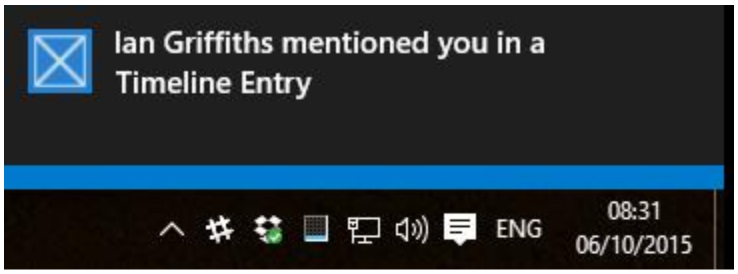
46. Right click on the **Rg.Api** project, choose **Publish**, and then click **Publish** to deploy this new code to your Azure API.
47. Next, set the client app as the startup project, run it.
48. Click the **Log In** button and log into the application.
49. Click the **Register for Notifications** button.

50. Wait for the dialog confirming registration (it can take 30 seconds or more), then close the client app.
51. Right-click on the **Rg.Web** project, choose **Publish**, and then click **Publish** to deploy the new notification handling code to your Web App. Remember, the **Rg.ServiceOperations** code runs in both your API and your Web App.
52. When the web app appears, log in.
53. Expand the menu and click on **Profile**. Enable device notifications.



☑Send my devices notifications when I'm mentioned

54. Expand the menu again and click **Home**. Type **@** in the message entry text box, select yourself from the user list, and then post the message. You should see a toast notification pop up.



Ian Griffiths mentioned you in a Timeline Entry

This verifies that notifications can be delivered even when your app is not running.

Let's review what you have done. You arranged for your client app to be able to receive push notifications and configured an Azure Notification Hub with the details it needs to be able to send those messages. The same hub could also be configured with similar details for an iOS app or an Android app had you written such apps. You added some code to your back end to enable the client app to supply its push registration details and to get those details to the notification hub. Having done that, raising notifications becomes very straightforward—you just ask the hub to send a message to all devices registered to a particular user id.

You're done! Pat yourself on the back for making it through.

Don't forget to commit your code before you wrap up.