Azure Fast Start for Mobile Application Development

Module 02: Microsoft Azure NotificationHub

Student Lab Manual

Instructor Edition (Book Title Hidden Style)

Version 1.5

**Conditions and Terms of Use**

Microsoft Confidential

This training package is proprietary and confidential, and is intended only for uses described in the training materials. Content and software is provided to you under a Non-Disclosure Agreement and cannot be distributed. Copying or disclosing all or any portion of the content and/or software included in such packages is strictly prohibited.

The contents of this package are for informational and training purposes only and are provided "as is" without warranty of any kind, whether express or implied, including but not limited to the implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

Training package content, including URLs and other Internet Web site references, is subject to change without notice. Because Microsoft must respond to changing market conditions, the content should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented after the date of publication. Unless otherwise noted, the companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, place, or event is intended or should be inferred.

© 2015 Microsoft Corporation. All rights reserved.

**Copyright and Trademarks**

© 2015 Microsoft Corporation. All rights reserved.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

For more information, see Use of Microsoft Copyrighted Content at  
<http://www.microsoft.com/en-us/legal/intellectualproperty/Permissions/default.aspx>

Microsoft, Internet Explorer, and Windows are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. Other Microsoft products mentioned herein may be either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. All other trademarks are property of their respective owners.

Contents

[Lab 1: Sending Simple Windows Notifications 6](#_Toc434331491)

[Exercise 1: Set up Your Environment 7](#_Toc434331492)

[Exercise 2: Send Your First Notification 19](#_Toc434331493)

[Exercise 3: Customize Windows 10 notifications 25](#_Toc434331494)

[Lab 2: Advanced Notification Scenarios 27](#_Toc434331495)

[Exercise 1: Target Devices Using Tags 28](#_Toc434331496)

[Exercise 2: Personalize Notifications with Push Variables 30](#_Toc434331497)

# Lab 1: Sending Simple Windows Notifications

#### Introduction

NotificationHub is a Microsoft Azure service that allows you to send notifications to any device.

Windows 10 brings those notifications to a new level by allowing advanced interactions within a notification.

#### Objectives

After this lab, you will be able to:

* Configure an application to receive notification.
* Send notifications from a back end.
* Display custom Windows 10 notifications.

#### Prerequisites

You must perform the following steps to prepare your computer for this lab:

* Execute the operations in Module 1 Lab: Azure Overview.
* Windows App Developer account.
* An Azure Account.

#### Estimated Time to Complete This Lab

45 minutes

#### Scenario

We want to integrate notifications to the Catalog Product Windows 10 application. The recommended architecture to send notifications implies a back end that registers and does send the notifications for the front end:

Device App: WAAD.POC.ProductCatalog.UWP

Backend:  
WAAD.POC.ProductCatalog.Backend

Azure: NotificationHub

WNS

## Exercise 1: Set up Your Environment

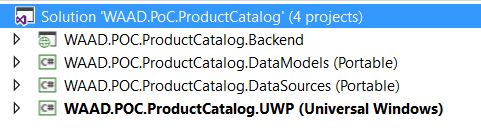
#### Objectives

In this exercise, you will:

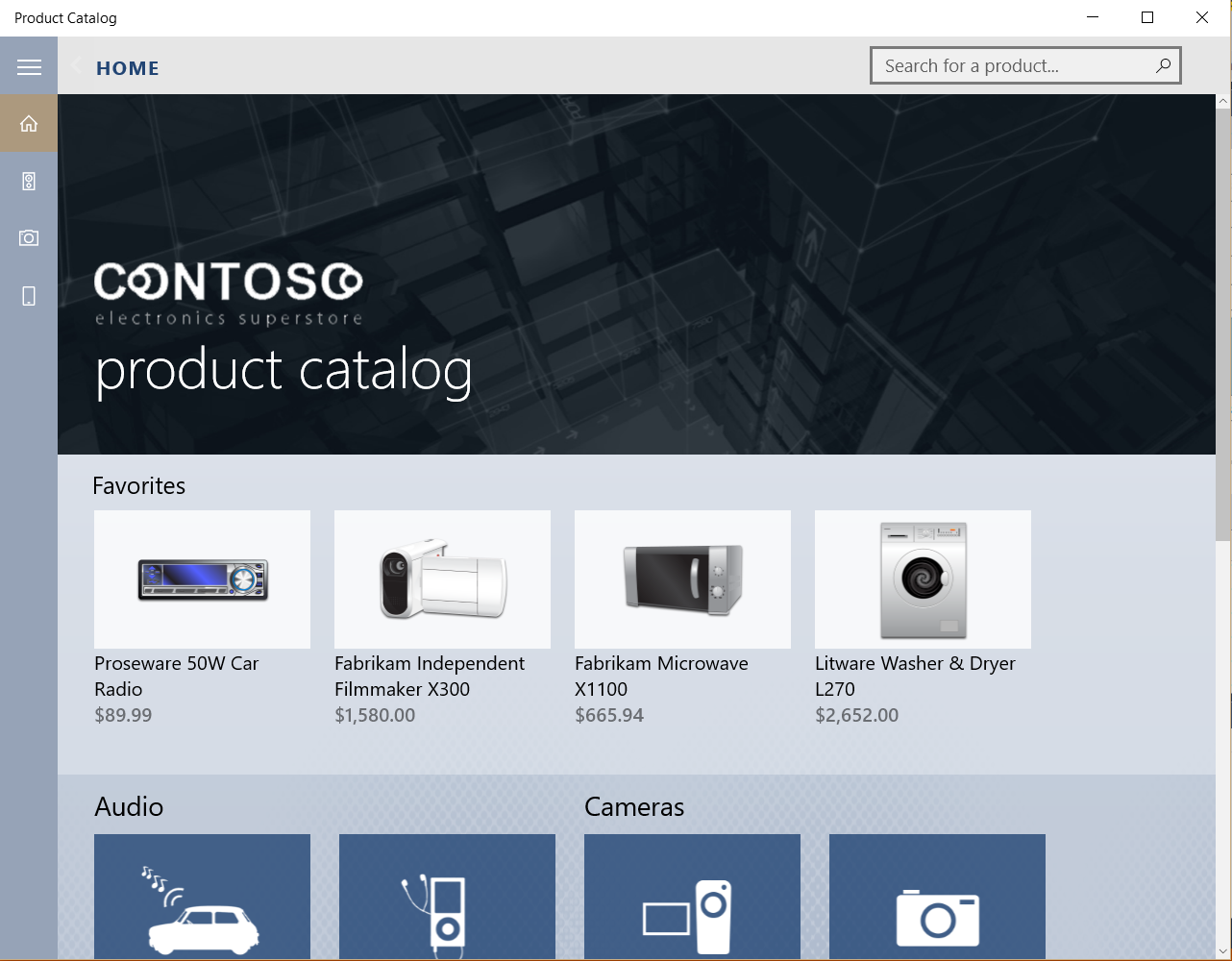
* Declare a new application on the Windows Dev Center.
* Configure the application to work with Microsoft Push Notification Service (MPNS) platform.
* Create an Azure NotificationHub.
* Send your first notification.

Task 1: Check the Solution

1. Open the solution file:   
   \Module02-NotificationHub\Start\WAAD.PoC.ProductCatalog.sln
2. Check all projects are loaded by looking at the Solution Explorer view.



1. Rebuild solution if necessary (**Right click on solution / Rebuild Solution**) and set as Startup project the WAAD.POC.ProductCatalog.UWP project (**Right click on the project / Set as StartUp Project**)
2. Check the application builds and launches by pressing the F5 key. Following app should start:

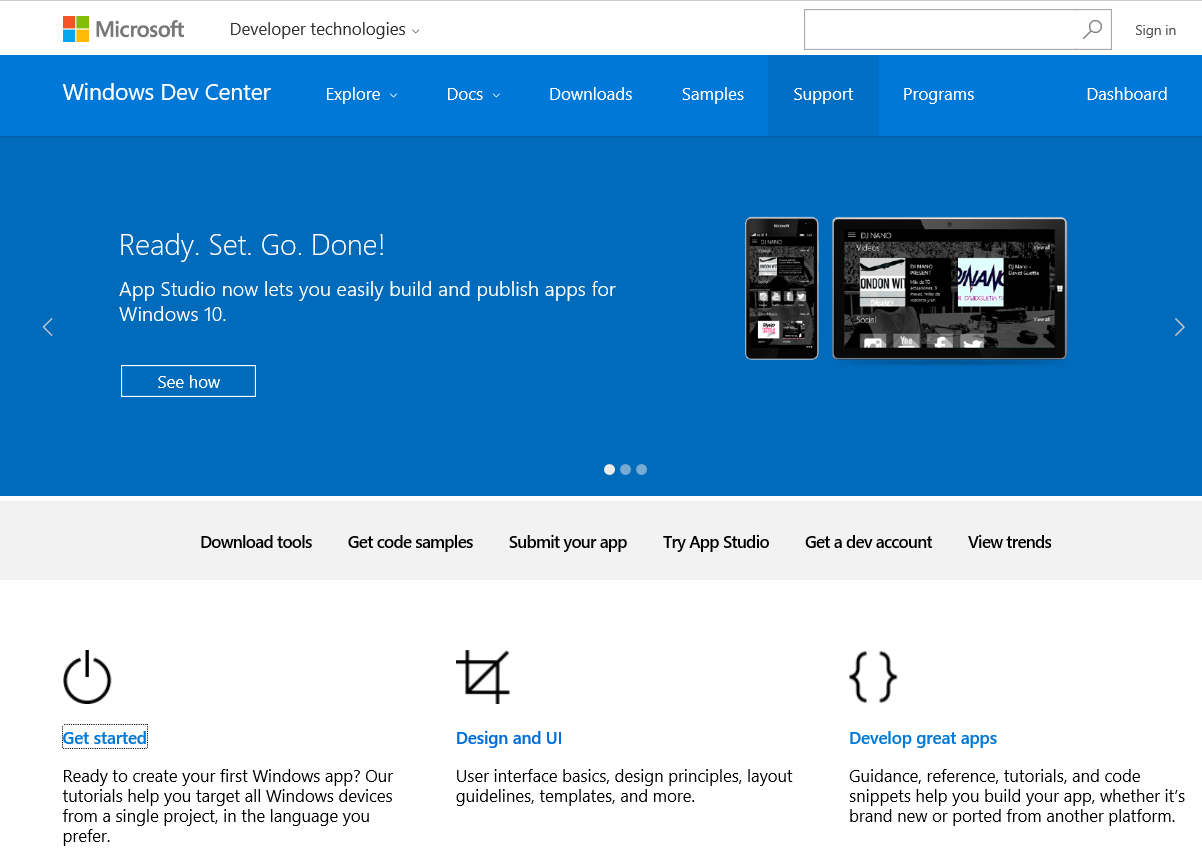


You can now close the app.

1. Now, set the WAAD.POC.ProductCatalog.Backend as startup project by right-clicking it and select **Set as Startup Project**.

Task 2: Declare Your App in the Windows Dev Center

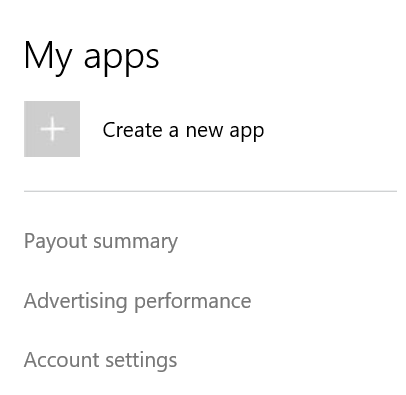
1. Go to the page <https://dev.windows.com/>
2. Click **Sign in**.



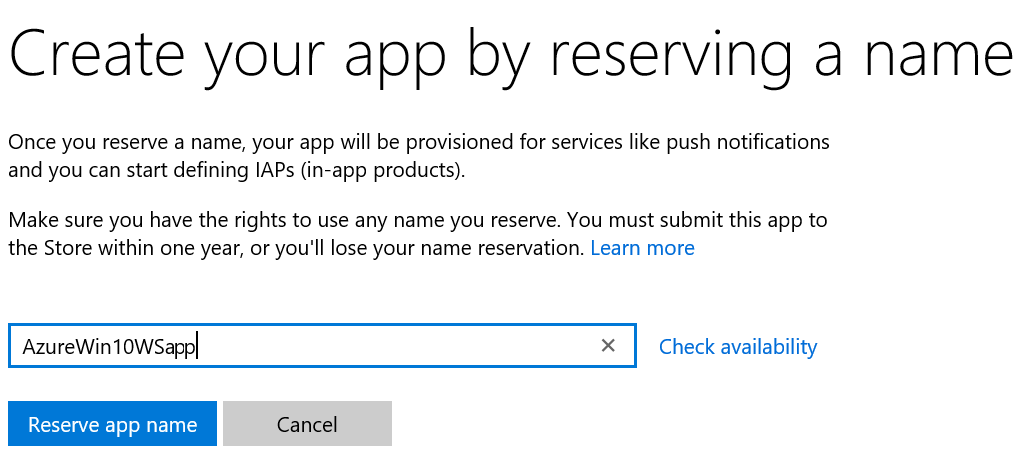
1. Now, click **Submit your app**.

**Note**: You may need to get a dev account first.

1. Select **Create a new app**.

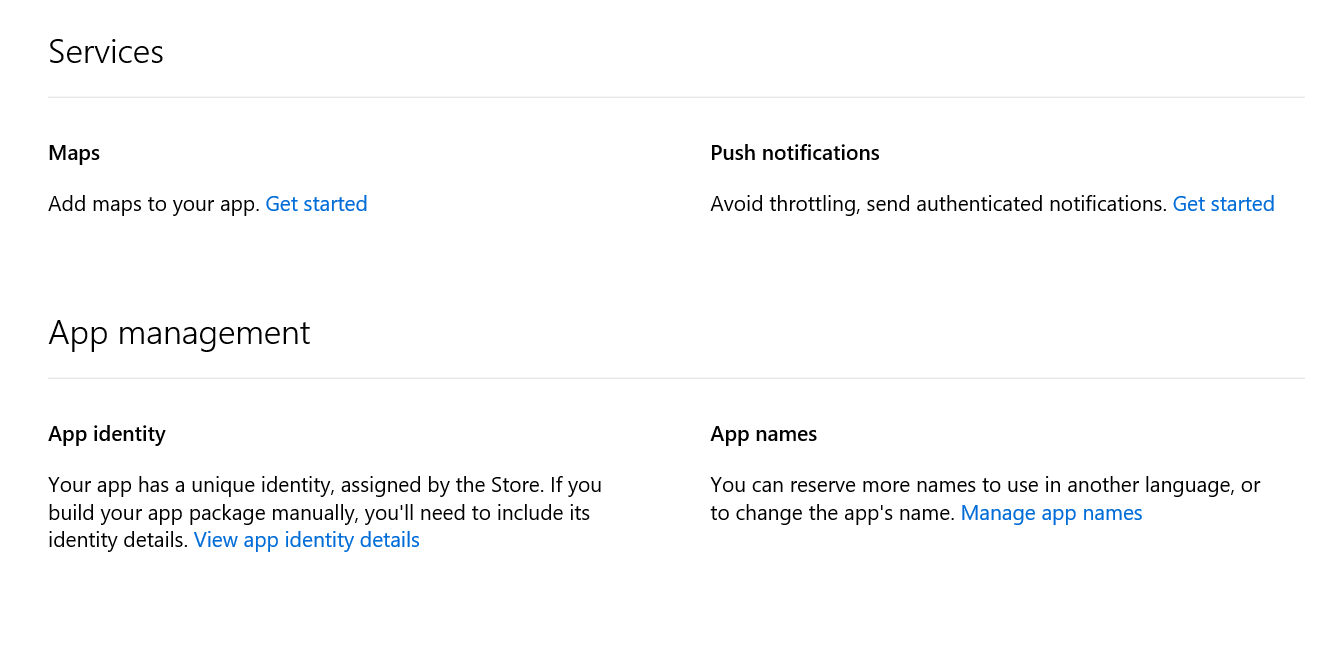


1. Choose a unique name for your app, click **Check availability**, and then **Reserve app name**.

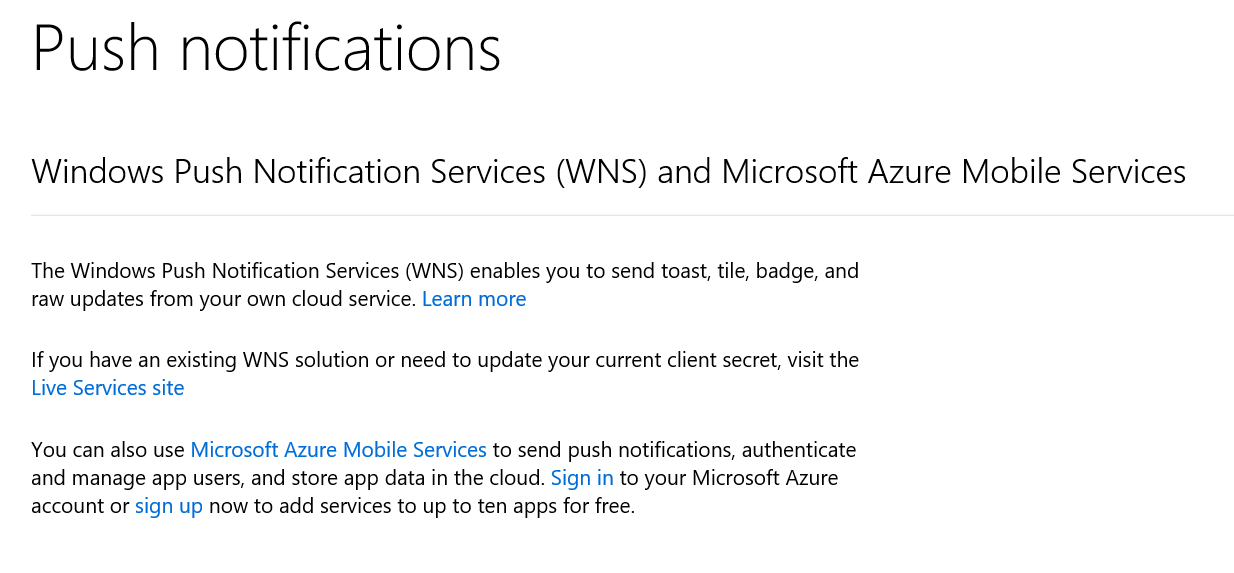


Task 2: Retrieve Your WNS SID and Secret

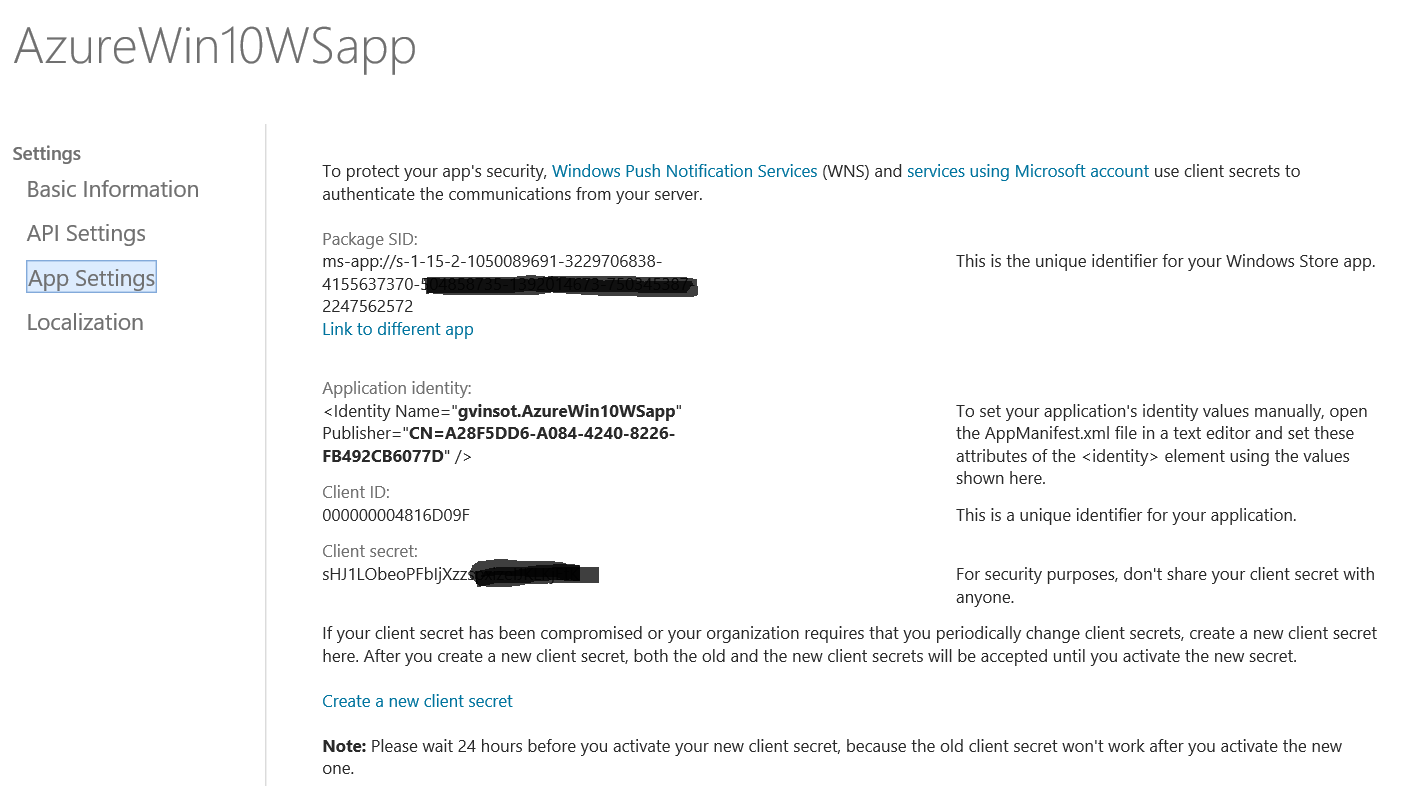
1. On the next page, scroll down, and then select **Get started** under the **Services / Push notifications** section.



1. To access your Windows Push Notification (WNS) information, click **Live Services site**.



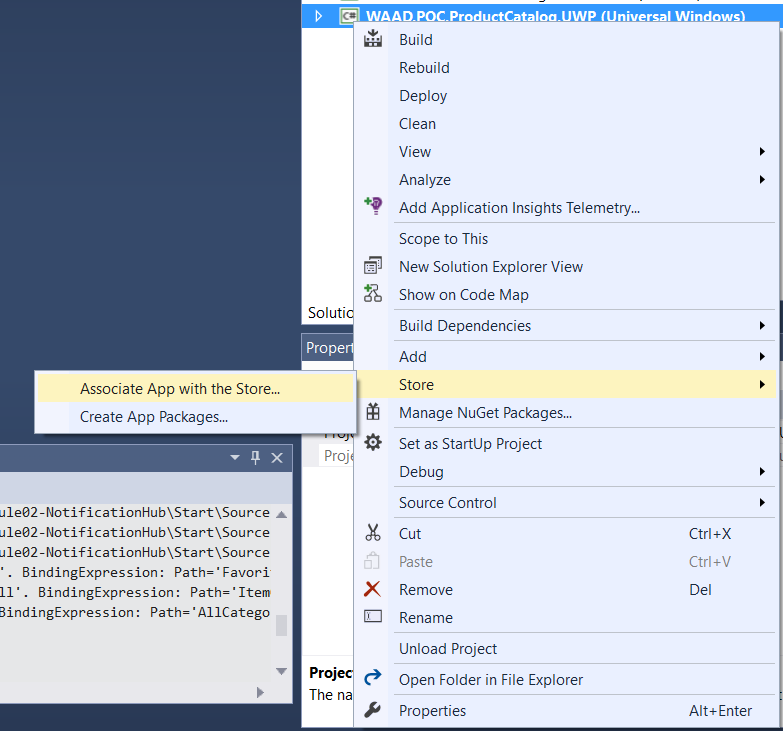
1. Copy your **Package SID** and **Client secret** information for later in a notepad:



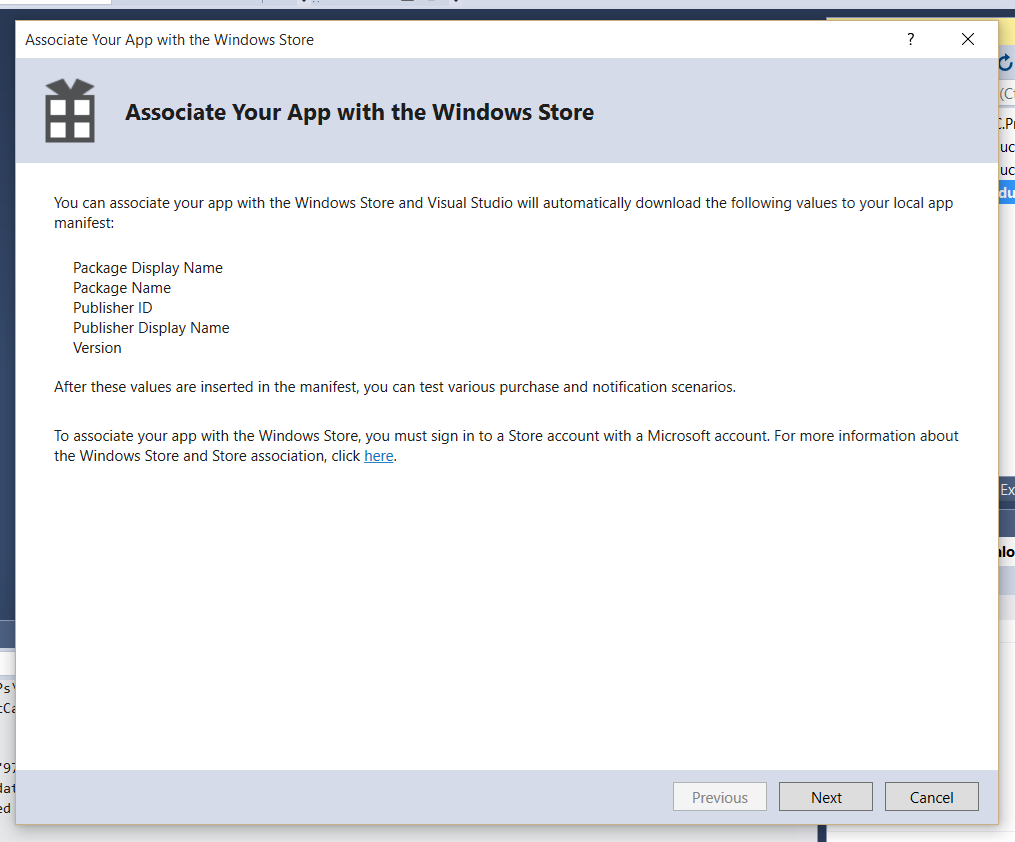
**Warning**: The client secret and package SID are important security credentials. Do not share these values with anyone or distribute them with your app.

Task 3: Associate Your Visual Studio Project to the Dev Center App

1. Return to your Visual Studio solution.
2. In the **Solution Explorer** view, right-click the project **WAAD.POC.ProductCatalog.UWP**, and then select **Store** > **Associate App with the Store**.

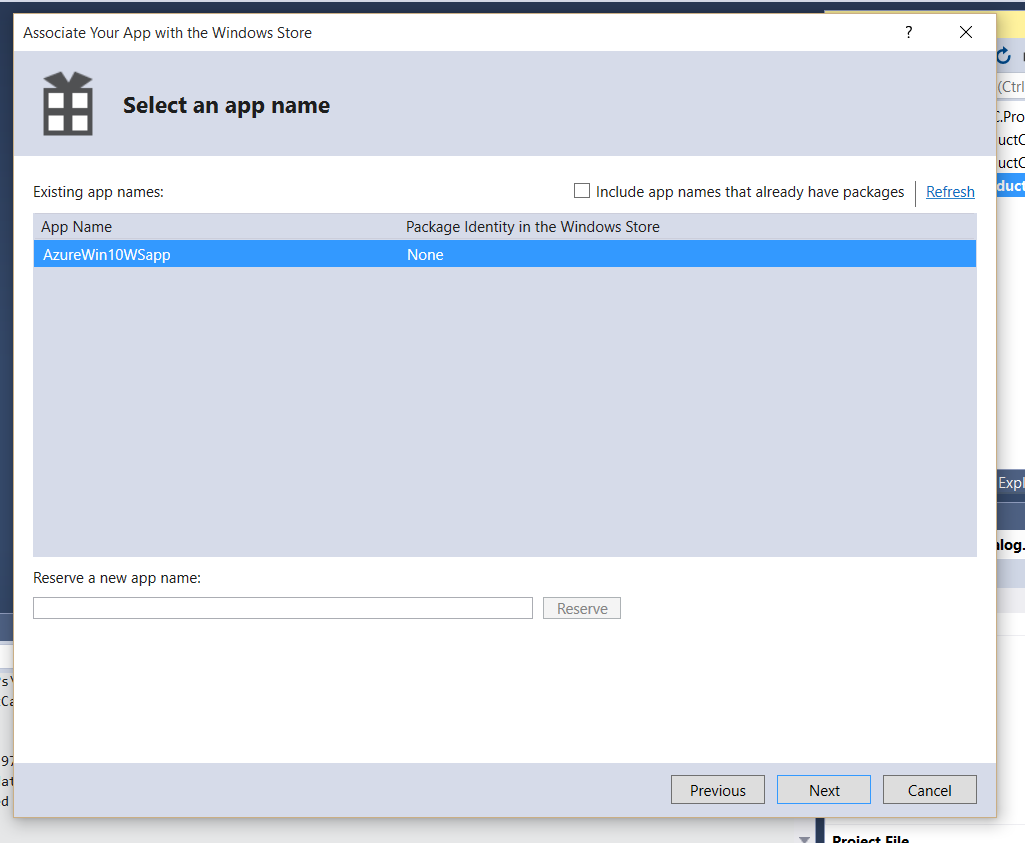


1. Click **Next** and enter your Dev Center logon information.

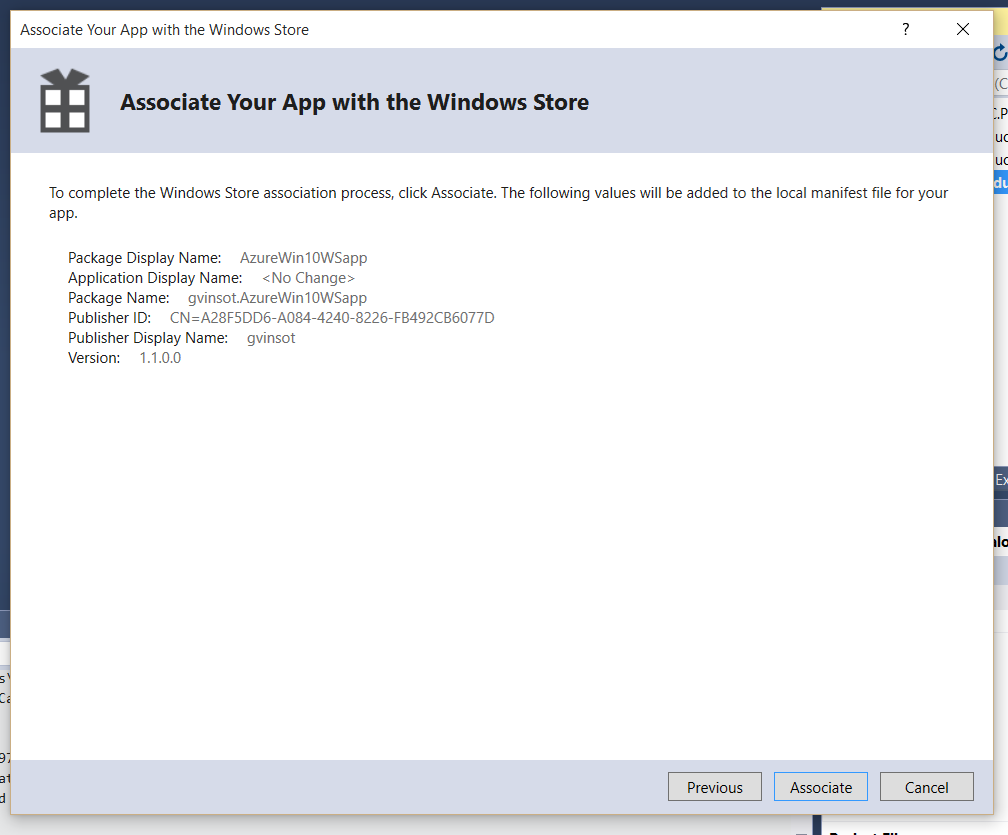


1. You should see following list of your registered applications.

Select your **Application** and click **Next**.



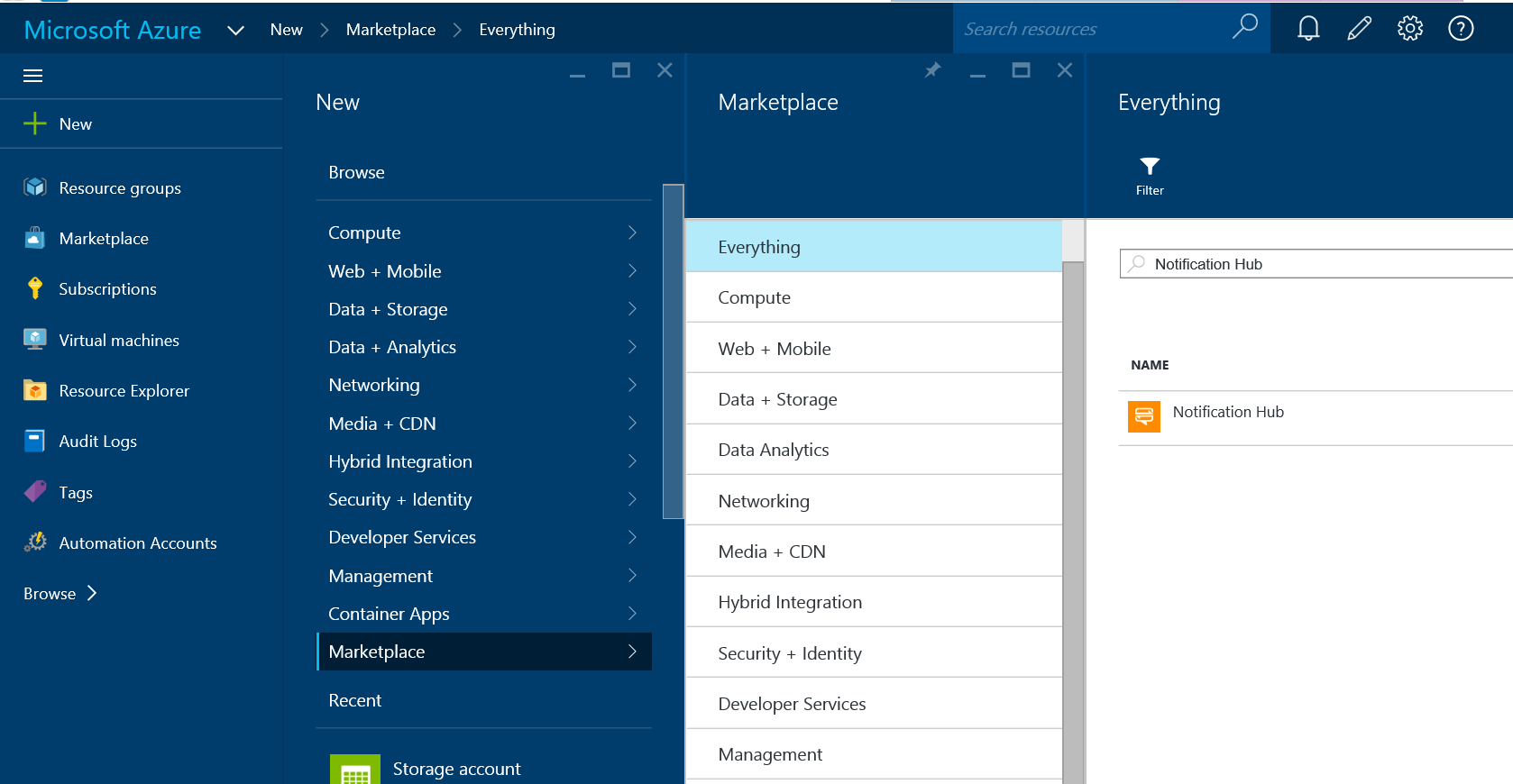
1. To associate your project with your Dev Center App, click **Associate**.



Task 4: Create an Azure Notification Hub

The following steps demonstrate how to create a Notification hub service and register your Windows Notification Service into it.

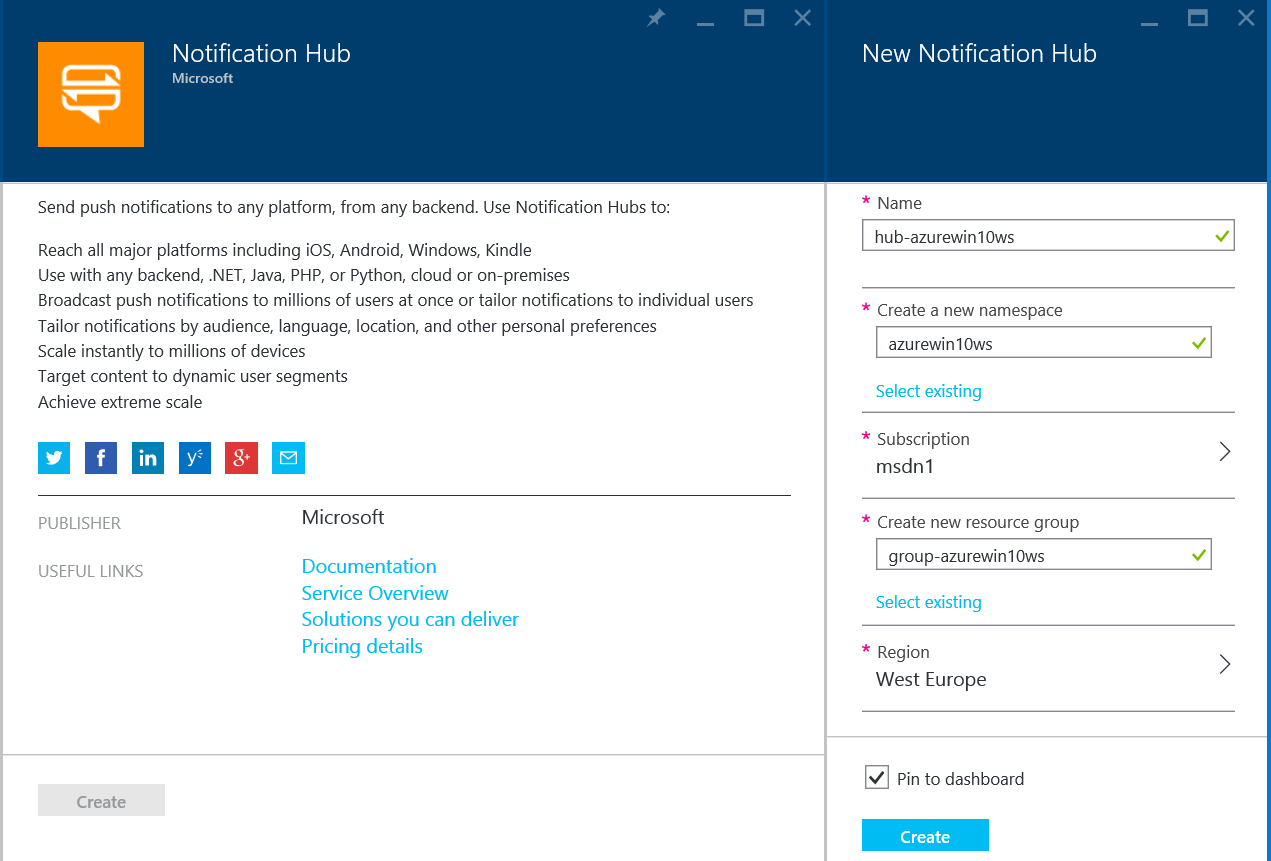
1. Go to the Azure Portal: <https://portal.azure.com/>
2. Authenticate.
3. Click **+NEW** > **Marketplace** > **Everything**, and search for **Notification Hub** > click the **Notification Hub** item.



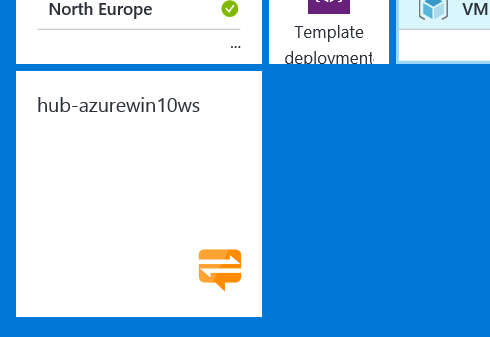
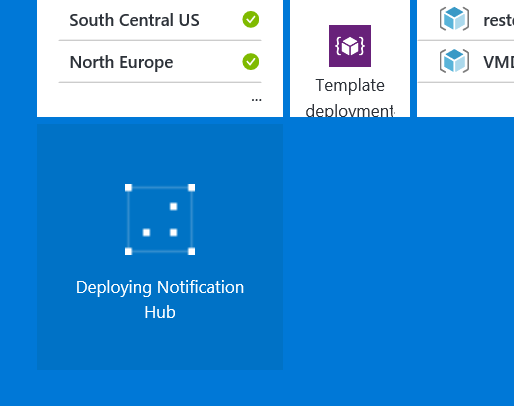
1. From the blade **Notification Hub**, you have access to all documentation and discussions about the service. Click **Create** and then set your service properties.

* Name: choose a name for your service.
* Subscription: select a subscription.
* Give a name for the namespace of your hub.
* Give a name for the resource group that will contain the service.
* Select a region.

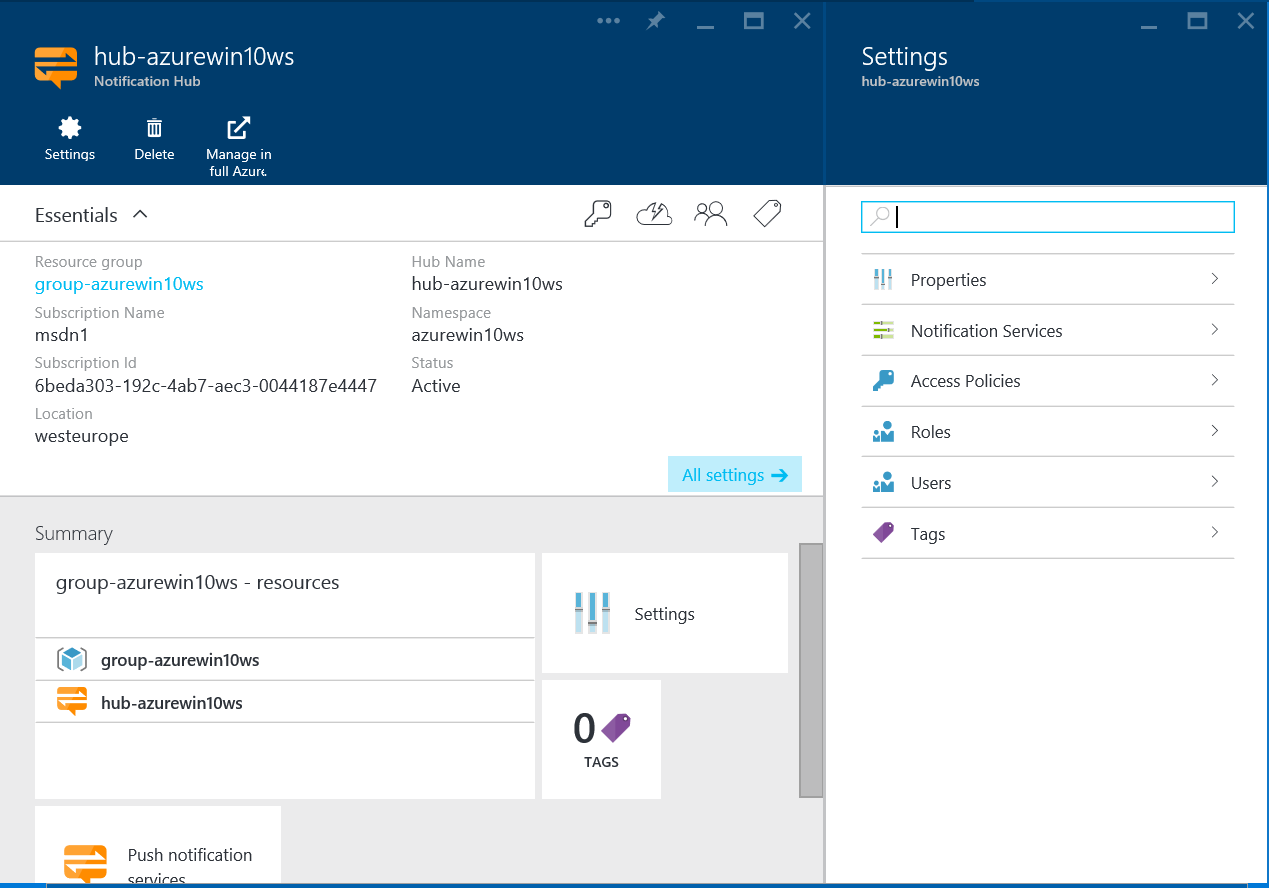
1. Finally, click **Create**.



1. Wait (1 minute) for the service to be created.

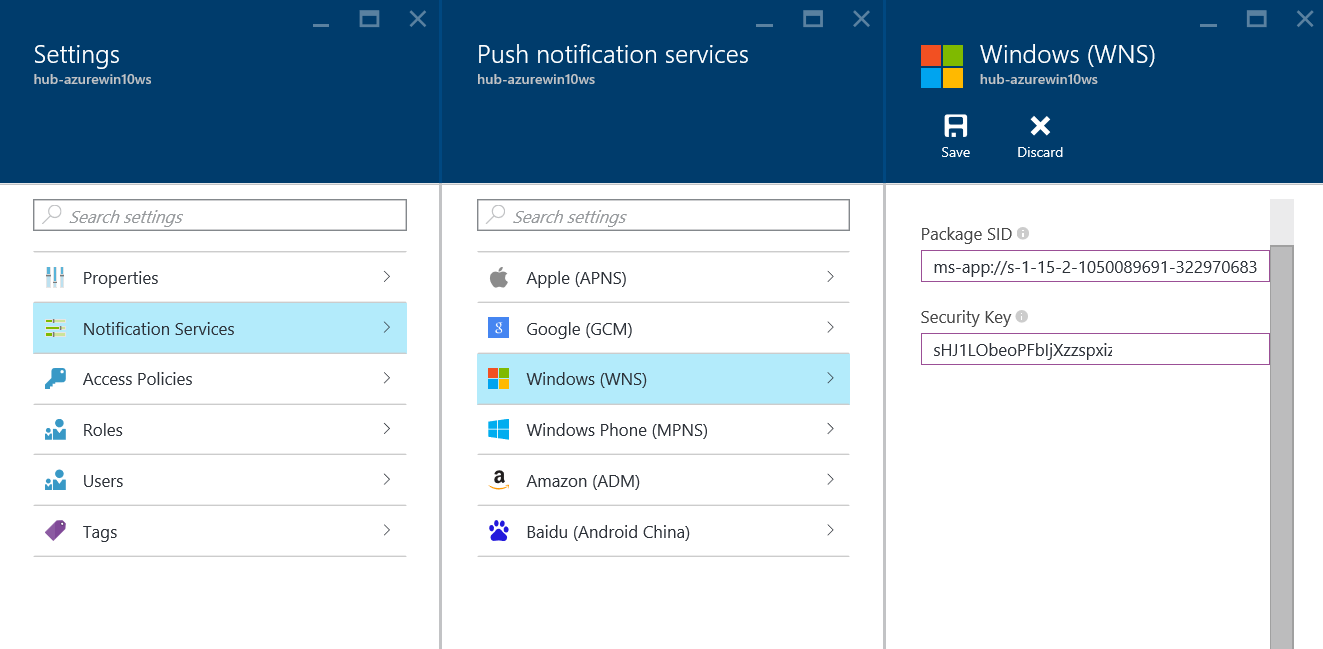
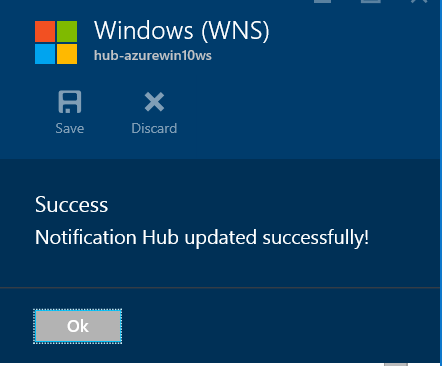


1. If the blade does not open automatically, click your **hub service tile**. You should see the following blade:



1. Select **Notification Services** in the Settings blade and then **Windows (WNS)**.

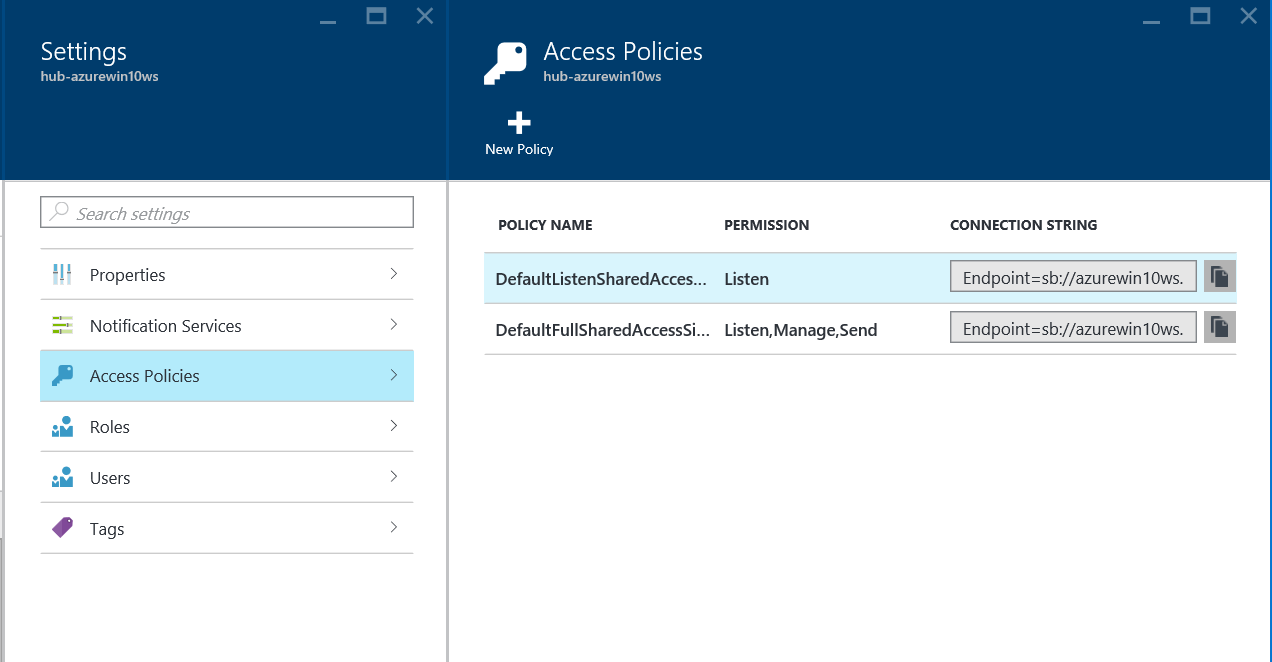
You must enter here the Package SID and Security Key you retrieved in last step of Task 2.



You have finished creating your Azure Notification Hub Service.

1. Now, retrieve your connection string by clicking **Access Policies** in the **Settings** blade.

Copy the DefaultFullSharedAccessSignature and keep it somewhere for later.

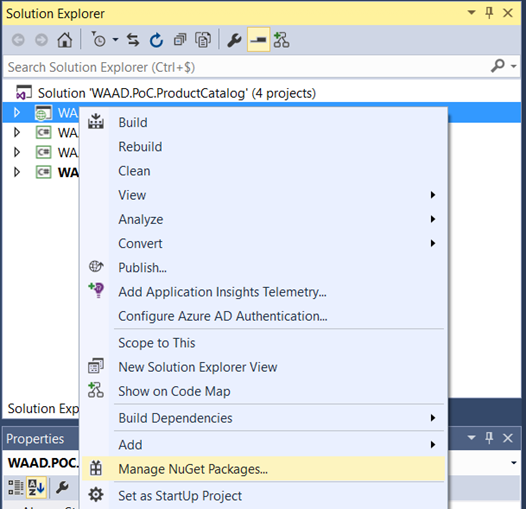


## Exercise 2: Send Your First Notification

Task 1: Add the NuGet Package

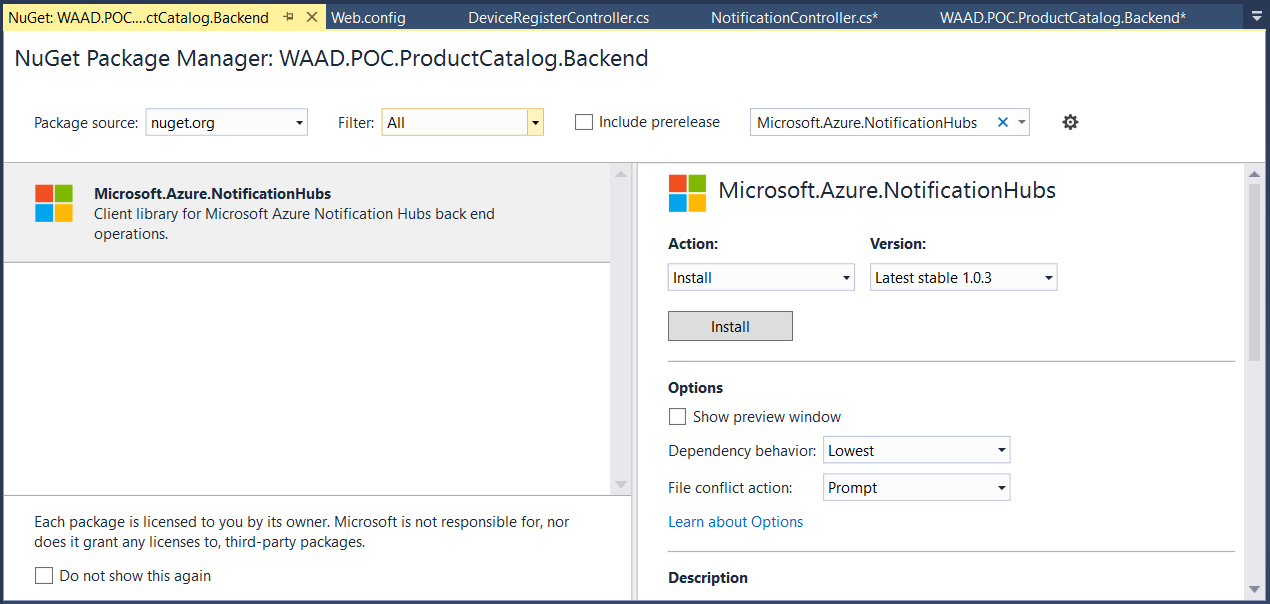
This back end will be responsible for registering sending notifications.

1. Return to your Visual Studio solution.
2. In the **Solution Explorer**, right-click the project **WAAD.POC.ProductCatalog.Backend**, and then click **Manage NuGet Packages**.



1. Search for **Microsoft.Azure.NotificationHubs** in the nuget.org Package source.

Select **Microsoft.Azure.NotificationHubs** item and click **Install**.



1. Read agreement and then click **I accept**.

Task 2: Set the Connection String and Hub Name

1. Expand the **WAAD.POC.ProductCatalog.Backend** project, and open the file **Web.Config**.
2. In the **appSettings** section, replace the value with the connection string you retrieved from the Azure Portal.

<appSettings>

<add key="NotificationHubConnectionString" value="ENTER YOUR CONNECTION STRING HERE" />

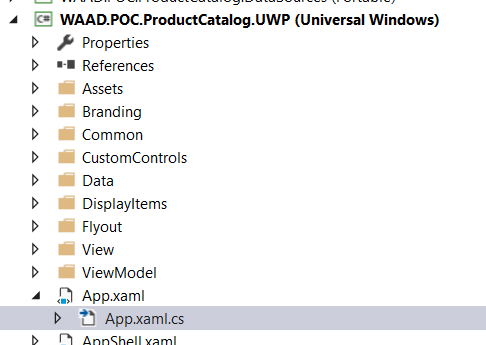
<add key="NotificationHubName" value="ENTER THE NAME OF YOUR NOTIFICATION HUB" />

</appSettings>

Task 3: Code the Device Registration

When your application starts, it will need to register itself to the WNS platform.

1. Open the project **WAAD.POC.ProductCatalog.UWP**.
2. Expand file **App.xaml**.
3. Open file **App.xaml.cs**.



1. Add following declaration at the top:

using Windows.Networking.PushNotifications;

1. At the end of the method OnLaunched(LaunchActivatedEventArgs e)

Add following lines to get the WNS channel and push it to the back end:

var channel = await PushNotificationChannelManager.CreatePushNotificationChannelForApplicationAsync();

HttpClient client = new HttpClient();

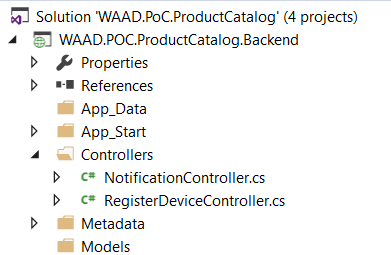
// Encode to Base64 to prevent alteration by the server side WebAPI code

string Base64EncodedChannelUri = Convert.ToBase64String(Encoding.UTF8.GetBytes(channel.Uri));

var postContent = new StringContent("=" + Base64EncodedChannelUri, Encoding.UTF8, "application/x-www-form-urlencoded");

var response = await client.PostAsync("http://localhost:27362/api/RegisterDevice", postContent);

1. Expand the project **WAAD.POC.ProductCatalog.Backend**.
2. Open the file **RegisterDeviceController.cs**.



1. Add following declaration at the top:

using Microsoft.Azure.NotificationHubs;

1. Add following lines in the Post method to register the device:

var connectionString = ConfigurationManager.AppSettings["NotificationHubConnectionString"];

var notificationHubName = ConfigurationManager.AppSettings["NotificationHubName"];

NotificationHubClient hub = NotificationHubClient.CreateClientFromConnectionString(connectionString, notificationHubName);

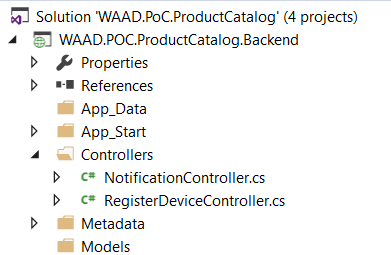
// Decode from Base64

string Base64DecodedChannelUri = Encoding.UTF8.GetString(Convert.FromBase64String(value));

hub.DeleteRegistrationsByChannelAsync(Base64DecodedChannelUri).Wait();

hub.CreateWindowsNativeRegistrationAsync(Base64DecodedChannelUri);

Task 4: Make Your Back End Ready to Send Notifications



1. Open the file **NotificationController.cs**.
2. Add following declaration at the top:

using Microsoft.Azure.NotificationHubs;

1. Add following lines in the Get([FromUri]string message) method to register the device:

string payloadMessage = String.Format(@"<toast><visual><binding template=""ToastText01""><text id=""1"">{0}</text></binding></visual></toast>",message);

var connectionString = ConfigurationManager.AppSettings["NotificationHubConnectionString"];

var notificationHubName = ConfigurationManager.AppSettings["NotificationHubName"];

NotificationHubClient hub = NotificationHubClient.CreateClientFromConnectionString(connectionString, notificationHubName);

var result = await hub.SendWindowsNativeNotificationAsync(payloadMessage);

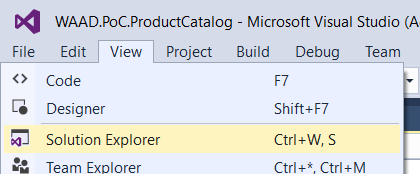
HttpResponseMessage response = Request.CreateResponse(HttpStatusCode.OK);

response.Content = new StringContent("Success");

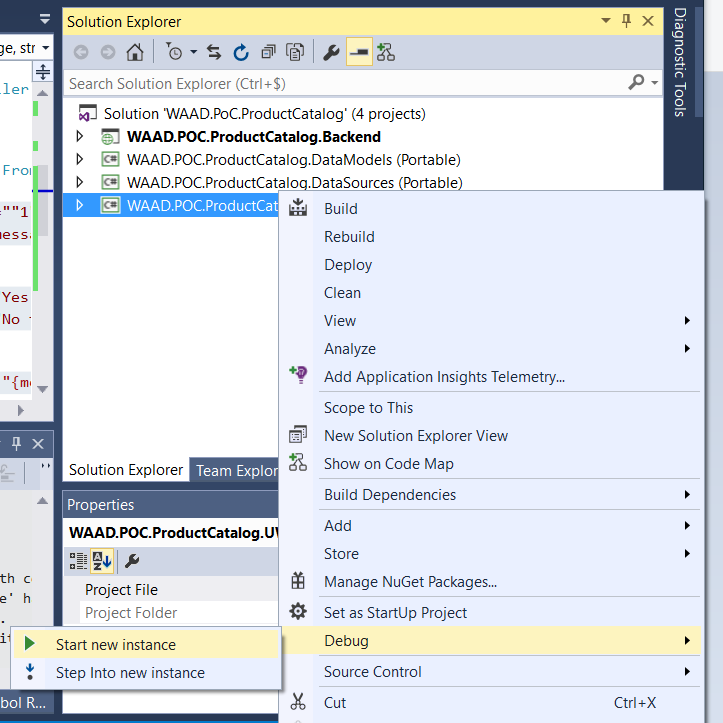
return response;

Task 5: Launch Your App and Send a Notification

1. Launch the back end by pressing F5.
2. You may need to make the Solution Explorer reappear by going in the menu **View** > **Solution Explorer**.



1. From the **Solution Explorer**, launch the application **WAAD.POC.ProductCatalog.UWP**   
   by right-clicking the **Project** > **Debug** > **Start new instance**.

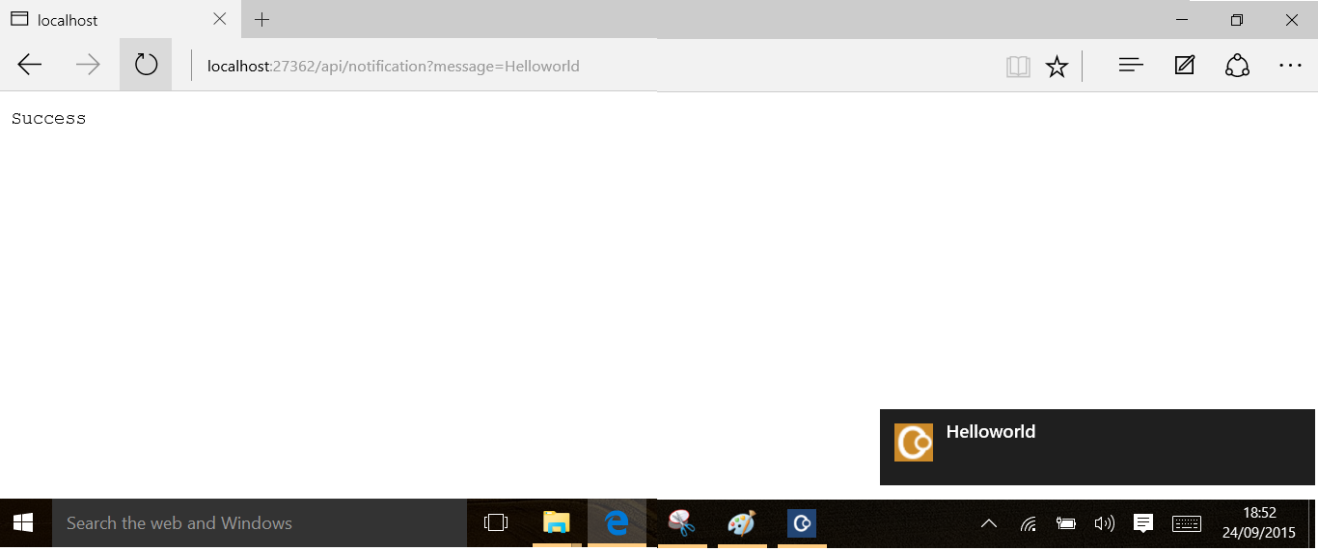


1. You will now request the back end to send a notification.

Start Internet Explorer or Edge and call the following URL:

<http://localhost:27362/api/notification?message=Helloworld>

You should receive the notification:



**Warning**: If you are running on battery and you have less than 20 percent left, by default, notification are shut down to save battery.

## Exercise 3: Customize Windows 10 notifications

Using Windows 10, you have new notifications possibilities. In this exercise, you will create a custom toast notification with an integrated interaction.

Task 1: Modify the Notification Payload

Modify the payload to show buttons in a generic toast.

1. Open the file **NotificationController.cs**.
2. Change the payload messageTemplate variable by replacing it with this:

string payloadMessage = String.Format(@"<toast launch=""1""><visual><binding template=""ToastGeneric"">

<text id=""1"">Dear customer, Get {0} if you come visit us today ! Best Regards from our team.</text>

</binding></visual>

<actions>

<action activationType=""foreground"" content=""Yes I will !"" arguments=""Yes""/>

<action activationType=""background"" content=""No thanks."" arguments=""No""/>

</actions>

</toast>",message);

Task 2: Modify Your App to Receive Toast Interaction Values

1. Expand the project **WAAD.POC.ProductCatalog.UWP** and open the file **App.xaml.cs**.
2. Add the following method to the class. This method will receive interaction message from the toast message.

protected override void OnActivated(IActivatedEventArgs args)

{

base.OnActivated(args);

if(args as ToastNotificationActivatedEventArgs != null)

{

string userChoice = (args as ToastNotificationActivatedEventArgs).Argument;

var dialog = new Windows.UI.Popups.MessageDialog("User selected : "+userChoice);

dialog.ShowAsync();

}

}

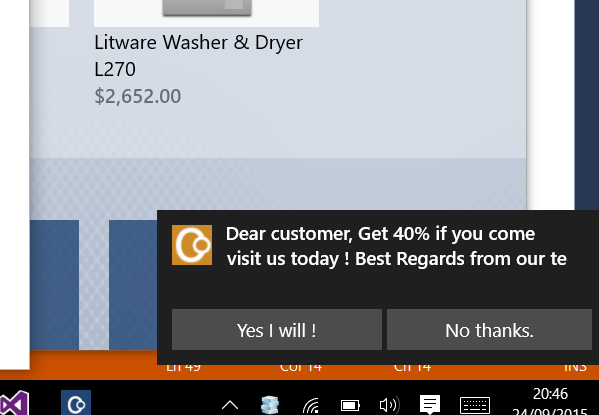
Task 3: Launch the App

1. Rebuild solution (**Right click on solution / Rebuild Solution**)
2. Launch the back end by pressing **F5**.
3. From the Solution Explorer, launch the application WAAD.POC.ProductCatalog.UWP   
   by right-clicking the **Project** > **Debug** > **Start new instance**.
4. You will now tell the back end to send a notification.

Start Internet Explorer or Edge and call the following URL:

[http://localhost:27362/api/notification?message=40%](http://localhost:27362/api/notification?message=40%25)

1. When the notification appears, click **Yes I will !**



1. The application opens up a message box indicating it received information from the notification.



# Lab 2: Advanced Notification Scenarios

#### Introduction

We are going to make your solution send targeted and personalized notifications.

#### Objectives

After this lab, you will be able to:

* Send targeted notifications.
* Personalize notifications.

#### Prerequisites

You must perform the following steps to prepare your computer for this lab:

* Execute the operations in Module 1 Lab: Azure Overview.
* Execute Lab 1 in this module.

#### Estimated Time to Complete This Lab

15 minutes

## Exercise 1: Target Devices Using Tags

#### Objectives

In this exercise, you will:

* Adapt the existing app to register with a tag.
* Send a targeted notification.

Task 1: Modify the Device Registration

Modify the registration to register the device for a tag.

1. Open the file **RegisterDeviceController.cs**.
2. Replace the line:

hub.CreateWindowsNativeRegistrationAsync(Base64DecodedChannelUri);

with this one:

hub.CreateWindowsNativeRegistrationAsync(Base64DecodedChannelUri,new List<string>{"Car","Microwave" });

Task 2: Modify the Notification to Target Tags

1. Open the file **NotificationController.cs**.
2. Change the declaration of the **Get** method to retrieve a tag variable:

public async Task<HttpResponseMessage> Get([FromUri]string message, [FromUri]string tag)

1. Change call to **SendWindowsNativeNotificationAsync** to take this tag into account:

var result = await hub.SendWindowsNativeNotificationAsync(payloadMessage,tag);

Task 3: Launch the App

1. Rebuild solution (**Right click on solution / Rebuild Solution**)
2. Launch the back end by pressing F5.
3. From the solution explorer, launch the application WAAD.POC.ProductCatalog.UWP   
   by right-clicking the **Project** > **Debug** > **Start new instance**.
4. You will now tell the back end to send a targeted notification.

Start Internet Explorer or Edge and call the following URL:

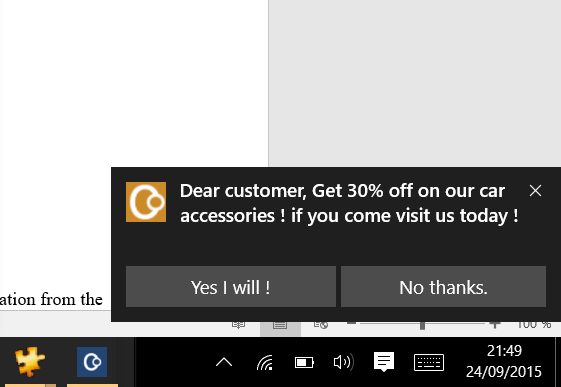
http://localhost:27362/api/notification?tag=Test&message=30%%20off%20on%20our%20car accessories%20!

* Nothing appears. It is normal because I registered my device for the tag **Car** and not **Test**.

1. You will now tell the back end to send a targeted notification to the right side tag.

Start Internet Explorer or Edge and call the following URL:

http://localhost:27362/api/notification?tag=Car&message=30%%20off%20on%20our%20car accessories%20!



## Exercise 2: Personalize Notifications with Push Variables

Task 1: Modify the Device Registration

Modify the registration to register push variables associated to the device.

By the time of writing this lab, the software development kits (SDK) still does not makes this feature avialable. So, we need to call Azure application programming interface (API) through a manual Representational State Transfer (REST) call.

To do that, just replace the file RegisterDeviceController.cs by the one in the folder Asset of the lab.

You can examine the file and see that we have push variables in the XML message:

<?xml version=""1.0"" encoding=""utf-8""?>

<entry xmlns=""http://www.w3.org/2005/Atom"">

<content type=""application/xml"">

<WindowsRegistrationDescription xmlns:i=""http://www.w3.org/2001/XMLSchema-instance"" xmlns=""http://schemas.microsoft.com/netservices/2010/10/servicebus/connect"">

<Tags>Car</Tags>

<PushVariables>{""userfirstname"":""John"",""userlastname"":""Smith""}</PushVariables>

<ChannelUri>{channeluri}</ChannelUri>

</WindowsRegistrationDescription>

</content>

</entry>

Task 2: Add Variables to Your Notification Payload

Modify the payload to show buttons in a generic toast.

1. Open the file **NotificationController.cs**.
2. Change the payload messageTemplate variable by replacing it with the following:

string payloadMessage = @"<toast launch=""1""><visual><binding template=""ToastGeneric"">

<text id=""1"">{'Dear '+ $(userfirstname) +', Get {message} if you come visit us today ! Best Regards from our team.'}</text>

</binding></visual>

<actions>

<action activationType=""foreground"" content=""Yes I will !"" arguments=""Yes""/>

<action activationType=""background"" content=""No thanks."" arguments=""No""/>

</actions>

</toast>";

payloadMessage = payloadMessage.Replace("{message}", message);

Task 3: Launch the App

1. Rebuild solution (**Right click on solution / Rebuild Solution**)
2. Launch the back end by pressing F5.
3. From the solution explorer, launch the application WAAD.POC.ProductCatalog.UWP   
   by right-clicking the **Project** > **Debug** > **Start new instance**.
4. You will now tell the back end to send a notification.

Start Internet Explorer or Edge and call the following URL:

http://localhost:27362/api/notification?tag=Car&message=30%%20off%20on%20our%20car accessories%20!

You should now receive your targeted personalized notification.