# Exercise 3: Using change notifications and track changes with Microsoft Graph

This tutorial teaches you how to build a .NET Core app that uses the Microsoft Graph API to receive notifications (webhooks) when a user account changes in Azure AD and perform queries using the delta query API to receive all changes to user accounts since the last query was made.

By the end of this exercise you will be able to:

* Monitor for changes using change notifications
* Get changes using a delta query

## Task 1: Create a new Azure AD web application

In this exercise, you will create a new Azure AD web application registration using the Azure AD admin center and grant administrator consent to the required permission scopes.

1. Open a browser and navigate to the Azure AD admin center [https://aad.portal.azure.com](https://aad.portal.azure.com/). Sign in using a **Work or School Account**.
2. Select **Azure Active Directory** in the leftmost navigation panel, then select **App registrations** under **Manage**.

* Screenshot of the App registrations
* Screenshot of the App registrations

1. Select **New registration** on the **Register an application** page.

* Screenshot of App Registrations page
* Screenshot of App Registrations page

1. Set the values as follows:
   * **Name:** Graph Notification Tutorial
   * **Supported account types**: Accounts in any organizational directory and personal Microsoft accounts
   * **Redirect URI**: Web > [http://localhost](http://localhost/)

* Screenshot of the Register an application page
* Screenshot of the Register an application page

1. Select **Register**.
2. On the **Graph Notification Tutorial** page, copy the value of the **Application (client) ID** and **Directory (tenant) ID;** save it, you will need them later in the tutorial.

* Screenshot of the application ID of the new app registration
* Screenshot of the application ID of the new app registration

1. Select **Manage > Certificates & secrets**.
2. Select **New client secret**.
3. Enter a value in **Description,** select one of the options for **Expires,** and select **Add**.

Screenshot of the Add a client secret dialog

Screenshot of the Add a client secret dialog

Screenshot of the Add a client secret dialog

Screenshot of the Add a client secret dialog

1. Copy the client secret value before you leave this page. You will need it later in the tutorial.

* **Important**: This client secret is never shown again, so make sure you copy it now.
* Screenshot of the newly added client secret
* Screenshot of the newly added client secret

1. Select **Manage** > **API Permissions**.
2. Select **Add a permission** and select **Microsoft Graph**.

* Screenshot selecting the Microsoft Graph service
* Screenshot selecting the Microsoft Graph service

1. Select **Application Permission**, expand the **User** group, and select **User.Read.All** scope.
2. Select **Add permissions** to save your changes.

* Screenshot selecting Microsoft Graph scop
* Screenshot selecting Microsoft Graph scop
* Screenshot of the newly added client secret
* Screenshot of the newly added client secret

1. The application requests an application permission with the **User.Read.All** scope. This permission requires administrative consent.
2. Select **Grant admin consent for Contoso**, then select **Yes** to consent this application, and grant the application access to your tenant using the scopes you specified.

Screenshot approved admin consent

Screenshot approved admin consent

## Task 2: Create .NET Core App

### Run ngrok

In order for the Microsoft Graph to send notifications to your application running on your development machine you need to use a tool such as ngrok to tunnel calls from the internet to your development machine. Ngrok allows calls from the internet to be directed to your application running locally without needing to create firewall rules.

1. Run ngrok by executing the following from the command line:

* ngrok http 5000

1. This will start ngrok and will tunnel requests from an external ngrok url to your development machine on port 5000. Copy the https forwarding address. In the example below that would be https://787b8292.ngrok.io. You will need this later.

### Create .NET Core WebApi App

1. Open your command prompt, navigate to a directory where you have rights to create your project, and run the following command to create a new .NET Core console application: dotnet new webapi -o msgraphapp
2. After creating the application, run the following commands to ensure your new project runs correctly:

* cd msgraphapp  
  dotnet add package Microsoft.Identity.Client  
  dotnet add package Microsoft.Graph  
  dotnet run

1. The application will start and output the following:

* info: Microsoft.Hosting.Lifetime[0]  
   Now listening on: https://localhost:5001  
  info: Microsoft.Hosting.Lifetime[0]  
   Now listening on: http://localhost:5000  
  info: Microsoft.Hosting.Lifetime[0]  
   Application started. Press Ctrl+C to shut down.  
  info: Microsoft.Hosting.Lifetime[0]  
   Hosting environment: Development  
  info: Microsoft.Hosting.Lifetime[0]  
   Content root path: [your file path]\msgraphapp

1. Stop the application running by pressing **CTRL+C**.
2. Open the application in Visual Studio Code using the following command: code .
3. If Visual Studio code displays a dialog box asking if you want to add required assets to the project, select **Yes**.
4. If Visual Studio Code displays a dialog box asking if you want to add required assets to the project, select **Yes**.

## Task 3: Code the HTTP API

Open the **Startup.cs** file and comment out the following line to disable ssl redirection.

//app.UseHttpsRedirection();

### Add model classes

The application uses several new model classes for (de)serialization of messages to/from the Microsoft Graph.

1. Right-click in the project file tree and select New Folder. Name it **Models**
2. Right-click the Models folder and add three new files:
   * **Notification.cs**
   * **ResourceData.cs**
   * **MyConfig.cs**
3. Replace the contents of **Notification.cs** with the following:

* using Newtonsoft.Json;  
  using System;  
  namespace msgraphapp.Models  
  {  
   public class Notifications  
   {  
   [JsonProperty(PropertyName = "value")]  
   public Notification[] Items { get; set; }  
   }  
   // A change notification.  
   public class Notification  
   {  
   // The type of change.  
   [JsonProperty(PropertyName = "changeType")]  
   public string ChangeType { get; set; }  
   // The client state used to verify that the notification is from Microsoft Graph. Compare the value received with the notification to the value you sent with the subscription request.  
   [JsonProperty(PropertyName = "clientState")]  
   public string ClientState { get; set; }  
   // The endpoint of the resource that changed. For example, a message uses the format ../Users/{user-id}/Messages/{message-id}  
   [JsonProperty(PropertyName = "resource")]  
   public string Resource { get; set; }  
   // The UTC date and time when the webhooks subscription expires.  
   [JsonProperty(PropertyName = "subscriptionExpirationDateTime")]  
   public DateTimeOffset SubscriptionExpirationDateTime { get; set; }  
   // The unique identifier for the webhooks subscription.  
   [JsonProperty(PropertyName = "subscriptionId")]  
   public string SubscriptionId { get; set; }  
   // Properties of the changed resource.  
   [JsonProperty(PropertyName = "resourceData")]  
   public ResourceData ResourceData { get; set; }  
   }  
  }

1. Replace the contents of **ResourceData.cs** with the following:

* using Newtonsoft.Json;  
  namespace msgraphapp.Models  
  {  
   public class ResourceData  
   {  
   // The ID of the resource.  
   [JsonProperty(PropertyName = "id")]  
   public string Id { get; set; }  
   // The OData etag property.  
   [JsonProperty(PropertyName = "@odata.etag")]  
   public string ODataEtag { get; set; }  
   // The OData ID of the resource. This is the same value as the resource property.  
   [JsonProperty(PropertyName = "@odata.id")]  
   public string ODataId { get; set; }  
   // The OData type of the resource: "#Microsoft.Graph.Message", "#Microsoft.Graph.Event", or "#Microsoft.Graph.Contact".  
   [JsonProperty(PropertyName = "@odata.type")]  
   public string ODataType { get; set; }  
   }  
  }

1. Replace the contents of **MyConfig.cs** with the following:

* namespace msgraphapp  
  {  
   public class MyConfig  
   {  
   public string AppId { get; set; }  
   public string AppSecret { get; set; }  
   public string TenantId { get; set; }  
   public string Ngrok { get; set; }  
   }  
  }

1. Open the **Startup.cs** file. Locate the method **ConfigureServices()** method and replace it with the following code:

* public void ConfigureServices(IServiceCollection services)  
  {  
   services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version\_3\_0);  
   var config = new MyConfig();  
   Configuration.Bind("MyConfig", config);  
   services.AddSingleton(config);  
  }

1. Open the **appsettings.json** file and replace the content with the following JSON.

* {  
   "Logging": {  
   "LogLevel": {  
   "Default": "Warning"  
   }  
   },  
   "MyConfig":  
   {  
   "AppId": "<APP ID>",  
   "AppSecret": "<APP SECRET>",  
   "TenantId": "<TENANT ID>",  
   "Ngrok": "<NGROK URL>"  
   }  
  }

1. Replace the following variables with the values you copied earlier:
   * should be set to the **https ngrok url** you copied earlier.
   * should be your **Office 365 tenant id** you copied earlier.
   * and should be the **application id** and **secret** you copied earlier when you created the application registration.

### Add notification controller

The application requires a new controller to process the subscription and notification. 1. Right-click the **Controllers** folder, select **New** **File**, and name the controller **NotificationsController.cs**.

1. Replace the contents of **NotificationController.cs** with the following code:

* using System;  
  using System.Collections.Generic;  
  using System.IO;  
  using System.Linq;  
  using System.Net.Http;  
  using System.Threading.Tasks;  
  using Microsoft.AspNetCore.Mvc;  
  using msgraphapp.Models;  
  using Newtonsoft.Json;  
  using System.Net;  
  using System.Threading;  
  using Microsoft.Graph;  
  using Microsoft.Identity.Client;  
  using System.Net.Http.Headers;  
  namespace msgraphapp.Controllers  
  {  
   [Route("api/[controller]")]  
   [ApiController]  
   public class NotificationsController : ControllerBase  
   {  
   private readonly MyConfig config;  
   public NotificationsController(MyConfig config)  
   {  
   this.config = config;  
   }  
   [HttpGet]  
   public async Task<ActionResult<string>> Get()  
   {  
   var graphServiceClient = GetGraphClient();  
   var sub = new Microsoft.Graph.Subscription();  
   sub.ChangeType = "updated";  
   sub.NotificationUrl = config.Ngrok + "/api/notifications";  
   sub.Resource = "/users";  
   sub.ExpirationDateTime = DateTime.UtcNow.AddMinutes(5);  
   sub.ClientState = "SecretClientState";  
   var newSubscription = await graphServiceClient  
   .Subscriptions  
   .Request()  
   .AddAsync(sub);  
   return $"Subscribed. Id: {newSubscription.Id}, Expiration: {newSubscription.ExpirationDateTime}";  
   }  
   public async Task<ActionResult<string>> Post([FromQuery]string validationToken = null)  
   {  
   // handle validation  
   if(!string.IsNullOrEmpty(validationToken))  
   {  
   Console.WriteLine($"Received Token: '{validationToken}'");  
   return Ok(validationToken);  
   }  
   // handle notifications  
   using (StreamReader reader = new StreamReader(Request.Body))  
   {  
   string content = await reader.ReadToEndAsync();  
   Console.WriteLine(content);  
   var notifications = JsonConvert.DeserializeObject<Notifications>(content);  
   foreach(var notification in notifications.Items)  
   {  
   Console.WriteLine($"Received notification: '{notification.Resource}', {notification.ResourceData?.Id}");  
   }  
   }  
   return Ok();  
   }  
   private GraphServiceClient GetGraphClient()  
   {  
   var graphClient = new GraphServiceClient(new DelegateAuthenticationProvider((requestMessage) => {  
   // get an access token for Graph  
   var accessToken = GetAccessToken().Result;  
   requestMessage  
   .Headers  
   .Authorization = new AuthenticationHeaderValue("bearer", accessToken);  
   return Task.FromResult(0);  
   }));  
   return graphClient;  
   }  
   private async Task<string> GetAccessToken()  
   {  
   IConfidentialClientApplication app = ConfidentialClientApplicationBuilder.Create(config.AppId)  
   .WithClientSecret(config.AppSecret)  
   .WithAuthority($"https://login.microsoftonline.com/{config.TenantId}")  
   .WithRedirectUri("https://daemon")  
   .Build();  
   string[] scopes = new string[] { "https://graph.microsoft.com/.default" };  
   var result = await app.AcquireTokenForClient(scopes).ExecuteAsync();  
   return result.AccessToken;  
   }  
   }  
  }

1. **Save** all files.

## Task 4: Run the application

Update the Visual Studio debugger launch configuration:

**Note**: By default, the .NET Core launch configuration will open a browser and navigate to the default URL for the application when launching the debugger. For this application, we instead want to navigate to the NGrok URL. If you leave the launch configuration as is, each time you debug the application it will display a broken page. You can just change the URL, or change the launch configuration to not launch the browser:

1. In Visual Studio Code, open the file **.vscode/launch.json**.
2. Delete the following section in the default configuration:

* // Enable launching a web browser when ASP.NET Core starts. For more information: https://aka.ms/VSCode-CS-LaunchJson-WebBrowser  
  "serverReadyAction": {  
   "action": "openExternally",  
   "pattern": "^\\s\*Now listening on:\\s+(https?://\\S+)"  
  },

1. **Save** your changes.
2. In **Visual Studio Code**, select **Debug** > **Start debugging** to run the application. VS Code will build and start the application.
3. Once you see messages in the Debug Console window that states the application started proceed with the next steps.
4. Open a browser and navigate to **http://localhost:5000/api/notifications** to subscribe to change notifications. If successful you will see output that includes a subscription id.

* **Note**: Your application is now subscribed to receive notifications from the Microsoft Graph when an update is made on any user in the Office 365 tenant.

1. Trigger a notification:
   1. Open a browser and navigate to the **Microsoft 365 admin center** (https://admin.microsoft.com/AdminPortal).
   2. If prompted to login, sign-in using an admin account.
   3. Select **Users** > **Active users**.
2. Select an active user and select **Edit** for their **Contact information**.
3. Update the **Phone number** value with a new number and select **Save**.
4. In the **Visual Studio Code Debug Console**, you will see a notification has been received. Sometimes this may take a few minutes to arrive. An example of the output is below:

* Received notification: 'Users/7a7fded6-0269-42c2-a0be-512d58da4463', 7a7fded6-0269-42c2-a0be-512d58da4463

This indicates the application successfully received the notification from the Microsoft Graph for the user specified in the output. You can then use this information to query the Microsoft Graph for the users full details if you want to synchronize their details into your application.

## Task 5: Manage notification subscriptions

Subscriptions for notifications expire and need to be renewed periodically. The following steps will demonstrate how to renew notifications: ### Update Code

1. Open **Controllers > NotificationsController.cs** file
2. Add the following two member declarations to the **NotificationsController** class:

* private static Dictionary<string, Subscription> Subscriptions = new Dictionary<string, Subscription>();  
  private static Timer subscriptionTimer = null;

1. Add the following new methods. These will implement a background timer that will run every 15 seconds to check if subscriptions have expired. If they have, they will be renewed.

* private void CheckSubscriptions(Object stateInfo)  
  {  
   AutoResetEvent autoEvent = (AutoResetEvent)stateInfo;  
   Console.WriteLine($"Checking subscriptions {DateTime.Now.ToString("h:mm:ss.fff")}");  
   Console.WriteLine($"Current subscription count {Subscriptions.Count()}");  
   foreach(var subscription in Subscriptions)  
   {  
   // if the subscription expires in the next 2 min, renew it  
   if(subscription.Value.ExpirationDateTime < DateTime.UtcNow.AddMinutes(2))  
   {  
   RenewSubscription(subscription.Value);  
   }  
   }  
  }  
  private async void RenewSubscription(Subscription subscription)  
  {  
   Console.WriteLine($"Current subscription: {subscription.Id}, Expiration: {subscription.ExpirationDateTime}");  
   var graphServiceClient = GetGraphClient();  
   var newSubscription = new Subscription  
   {  
   ExpirationDateTime = DateTime.UtcNow.AddMinutes(5)  
   };  
   await graphServiceClient  
   .Subscriptions[subscription.Id]  
   .Request()  
   .UpdateAsync(newSubscription);  
   subscription.ExpirationDateTime = newSubscription.ExpirationDateTime;  
   Console.WriteLine($"Renewed subscription: {subscription.Id}, New Expiration: {subscription.ExpirationDateTime}");  
  }

1. The **CheckSubscriptions** method is called every 15 seconds by the timer. For production use this should be set to a more reasonable value to reduce the number of unnecessary calls to Microsoft Graph. The **RenewSubscription** method renews a subscription and is only called if a subscription is going to expire in the next two minutes.
2. Locate the method **Get()** and replace it with the following code:

* [HttpGet]  
  public async Task<ActionResult<string>> Get()  
  {  
   var graphServiceClient = GetGraphClient();  
   var sub = new Microsoft.Graph.Subscription();  
   sub.ChangeType = "updated";  
   sub.NotificationUrl = config.Ngrok + "/api/notifications";  
   sub.Resource = "/users";  
   sub.ExpirationDateTime = DateTime.UtcNow.AddMinutes(5);  
   sub.ClientState = "SecretClientState";  
   var newSubscription = await graphServiceClient  
   .Subscriptions  
   .Request()  
   .AddAsync(sub);  
   Subscriptions[newSubscription.Id] = newSubscription;  
   if (subscriptionTimer == null)  
   {  
   subscriptionTimer = new Timer(CheckSubscriptions, null, 5000, 15000);  
   }  
   return $"Subscribed. Id: {newSubscription.Id}, Expiration: {newSubscription.ExpirationDateTime}";  
  }

### Test the changes

1. Within Visual Studio Code, select Debug > Start debugging to run the application. Navigate to the following url: **http://localhost:5000/api/notifications**. This will register a new subscription.
2. In the Visual Studio Code Debug Console window, approximately every 15 seconds, notice the timer checking the subscription for expiration:

* Checking subscriptions 12:32:51.882  
  Current subscription count 1

1. Wait a few minutes and you will see the following when the subscription needs renewing:

* Renewed subscription: 07ca62cd-1a1b-453c-be7b-4d196b3c6b5b, New Expiration: 3/10/2019 7:43:22 PM +00:00

This indicates that the subscription was renewed and shows the new expiry time.

## Task 5: Query for changes

Microsoft Graph offers the ability to query for changes to a particular resource since you last called it. Using this option, combined with Change Notifications, enables a robust pattern for ensuring you don’t miss any changes to the resources. 1. Locate and open the following controller: **Controllers** > **NotificationsController.cs**. Add the following code to the existing **NotificationsControlle**r class.

```csharp  
private static object DeltaLink = null;  
private static IUserDeltaCollectionPage lastPage = null;  
private async Task CheckForUpdates()  
{  
 var graphClient = GetGraphClient();  
 // get a page of users  
 var users = await GetUsers(graphClient, DeltaLink);  
 OutputUsers(users);  
 // go through all of the pages so that we can get the delta link on the last page.  
 while (users.NextPageRequest != null)  
 {  
 users = users.NextPageRequest.GetAsync().Result;  
 OutputUsers(users);  
 }  
 object deltaLink;  
 if (users.AdditionalData.TryGetValue("@odata.deltaLink", out deltaLink))  
 {  
 DeltaLink = deltaLink;  
 }  
}  
private void OutputUsers(IUserDeltaCollectionPage users)  
{  
 foreach(var user in users)  
 {  
 var message = $"User: {user.Id}, {user.GivenName} {user.Surname}";  
 Console.WriteLine(message);  
 }  
}  
private async Task<IUserDeltaCollectionPage> GetUsers(GraphServiceClient graphClient, object deltaLink)  
{  
 IUserDeltaCollectionPage page;  
 if (lastPage == null)  
 {  
 page = await graphClient  
 .Users  
 .Delta()  
 .Request()  
 .GetAsync();  
 }  
 else  
 {  
 lastPage.InitializeNextPageRequest(graphClient, deltaLink.ToString());  
 page = await lastPage.NextPageRequest.GetAsync();  
 }  
 lastPage = page;  
 return page;  
}  
```

1. This code includes a new method, **CheckForUpdates(**), that will call the Microsoft Graph using the delta url and then pages through the results until it finds a new **deltalink** on the final page of results. It stores the url in memory until the code is notified again when another notification is triggered.
2. Locate the existing **Post()** method and replace it with the following code:

* public async Task<ActionResult<string>> Post([FromQuery]string validationToken = null)  
  {  
   // handle validation  
   if (!string.IsNullOrEmpty(validationToken))  
   {  
   Console.WriteLine($"Received Token: '{validationToken}'");  
   return Ok(validationToken);  
   }  
   // handle notifications  
   using (StreamReader reader = new StreamReader(Request.Body))  
   {  
   string content = await reader.ReadToEndAsync();  
   Console.WriteLine(content);  
   var notifications = JsonConvert.DeserializeObject<Notifications>(content);  
   foreach (var notification in notifications.Items)  
   {  
   Console.WriteLine($"Received notification: '{notification.Resource}', {notification.ResourceData?.Id}");  
   }  
   }  
   // use deltaquery to query for all updates  
   await CheckForUpdates();  
   return Ok();  
  }

1. The Post method will now call **CheckForUpdates** when a notification is received. **Save** all files.

### Test the changes

1. Within **Visual Studio Code**, select **Debug** > **Start debugging** to run the application. Navigate to the following url: **http://localhost:5000/api/notifications**. This will register a new subscription.
2. Open a browser and navigate to the **Microsoft 365 admin center** (**https://admin.microsoft.com/AdminPortal**).
3. If prompted to login, sign-in using an admin account.
   1. Select **Users > Active users**.
   2. Select an active **user** and select **Edit** for their **Contact** **information**.
   3. Update the **Mobile** **phone** value with a new number and select **Save**.
4. Wait for the notification to be received as indicated in the Visual Studio Code Debug Console:

* Received notification: 'Users/7a7fded6-0269-42c2-a0be-512d58da4463', 7a7fded6-0269-42c2-a0be-512d58da4463

1. The application will now initiate a delta query with the graph to get all the users and log out some of their details to the console output.

* User: 19e429d2-541a-4e0b-9873-6dff9f48fabe, Allan Deyoung  
  User: 05501e79-f527-4913-aabf-e535646d7ffa, Christie Cline  
  User: fecac4be-76e7-48ec-99df-df745854aa9c, Debra Berger  
  User: 4095c5c4-b960-43b9-ba53-ef806d169f3e, Diego Siciliani  
  User: b1246157-482f-420c-992c-fc26cbff74a5, Emily Braun  
  User: c2b510b7-1f76-4f75-a9c1-b3176b68d7ca, Enrico Cattaneo  
  User: 6ec9bd4b-fc6a-4653-a291-70d3809f2610, Grady Archie  
  User: b6924afe-cb7f-45a3-a904-c9d5d56e06ea, Henrietta Mueller  
  User: 0ee8d076-4f13-4e1a-a961-eac2b29c0ef6, Irvin Sayers  
  User: 31f66f05-ac9b-4723-9b5d-8f381f5a6e25, Isaiah Langer  
  User: 7ee95e20-247d-43ef-b368-d19d96550c81, Johanna Lorenz  
  User: b2fa93ac-19a0-499b-b1b6-afa76c44a301, Joni Sherman  
  User: 01db13c5-74fc-470a-8e45-d6d736f8a35b, Jordan Miller  
  User: fb0b8363-4126-4c34-8185-c998ff697a60, Lee Gu  
  User: ee75e249-a4c1-487b-a03a-5a170c2aa33f, Lidia Holloway  
  User: 5449bd61-cc63-40b9-b0a8-e83720eeefba, Lynne Robbins  
  User: 7ce295c3-25fa-4d79-8122-9a87d15e2438, Miriam Graham  
  User: 737fe0a7-0b67-47dc-b7a6-9cfc07870705, Nestor Wilke  
  User: a1572b58-35cd-41a0-804a-732bd978df3e, Patti Fernandez  
  User: 7275e1c4-5698-446c-8d1d-fa8b0503c78a, Pradeep Gupta  
  User: 96ab25eb-6b69-4481-9d28-7b01cf367170, Megan Bowen  
  User: 846327fa-e6d6-4a82-89ad-5fd313bff0cc, Alex Wilber  
  User: 200e4c7a-b778-436c-8690-7a6398e5fe6e, MOD Administrator  
  User: 7a7fded6-0269-42c2-a0be-512d58da4463, Adele Vance  
  User: 752f0102-90f2-4b8d-ae98-79dee995e35e, Removed?:deleted  
  User: 4887248a-6b48-4ba5-bdd5-fed89d8ea6a0, Removed?:deleted  
  User: e538b2d5-6481-4a90-a20a-21ad55ce4c1d, Removed?:deleted  
  User: bc5994d9-4404-4a14-8fb0-46b8dccca0ad, Removed?:deleted  
  User: d4e3a3e0-72e9-41a6-9538-c23e10a16122, Removed?:deleted

1. In the Microsoft 365 Admin Portal, repeat the process of editing a user and Save again.

The application will receive another notification and will query the graph again using the last delta link it received. However, this time you will notice that only the modified user was returned in the results.

User: 7a7fded6-0269-42c2-a0be-512d58da4463, Adele Vance

Using this combination of notifications with delta query you can be assured you won’t miss any updates to a resource. Notifications may be missed due to transient connection issues, however the next time your application gets a notification it will pick up all the changes since the last successful query.

## Review

You’ve now completed this exercise. In this exercise, you created.NET Core app that used the Microsoft Graph API to receive notifications (webhooks) when a user account changes in Azure AD and perform queries using the delta query API to receive all changes to user accounts since the last query was made.