Azure Developer Immersions

API Management

Azure provides two sets of services for Web APIs: API Apps and API Management. You're already using the first of these. Although you created a Web App and not an API App, these are really just very slightly different flavors of application in Azure App Service, and the full functionality of both Web Apps and API Apps are available from either kind of app. Therefore, you are already using the integration authentication support for API endpoints in your Web App, and you have supplied Azure with your Swagger metadata endpoint, making your Web App's API visible to the REST client code generation in Visual Studio. However, you are not yet using any API Management features.

API Management provides mechanisms to monitor and control access to an API, including the ability to prevent individual users from abusing the API by imposing rate limits. It supplies a Developer Portal for your API that developers who wish to use your application's services can log into to discover information about the API. It can consolidate multiple backend services behind a single public-facing API. It has a variety of options for authentication. For instance, it can impose various authentication requirements on API users, and it can also pass credentials through to the underlying service (which can make more sense for APIs such as ours, in which we already handle authentication). It also provides a Publisher Portal which you, as the owner of an API, can use to view analytics to discover how your API is being used.

There are four exercises in this walkthrough:

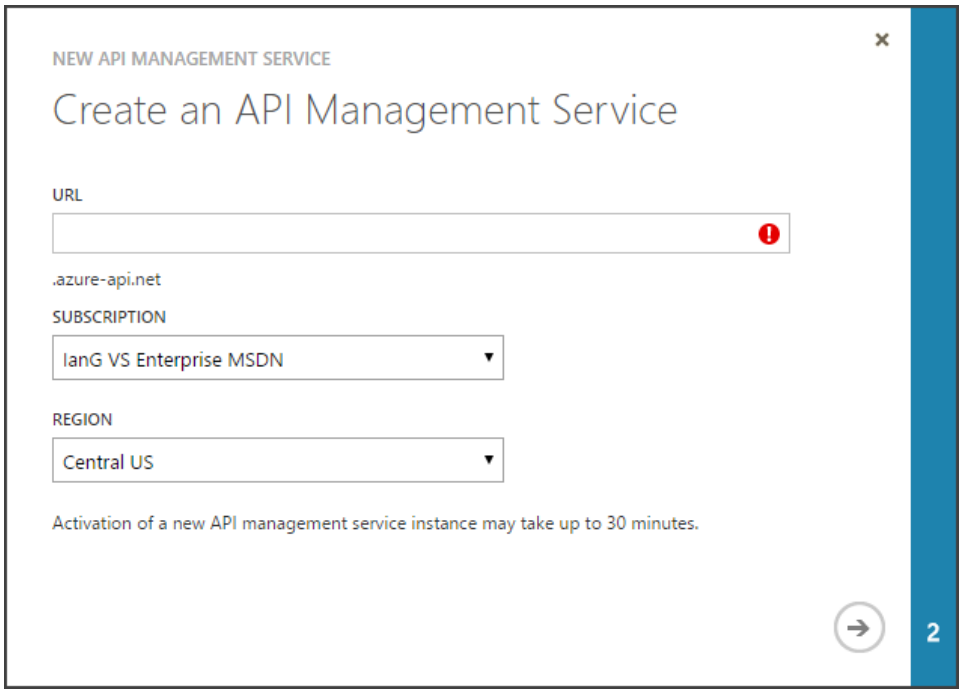
1. Create an API Management Instance
2. Wrap your API with API Management
3. Use the API Management Endpoint
4. Inspect the API Usage Information

Exercise 1: Create an API Management Instance

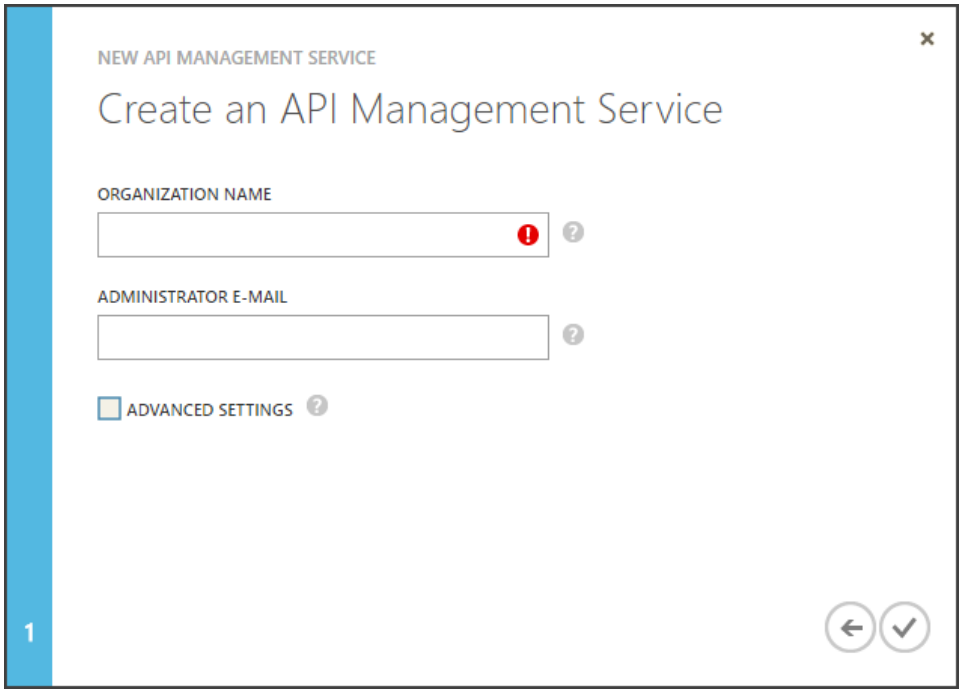
You will create a new API Management instance in Azure in this exercise.

> **Note:** This typically takes about half an hour for Azure to set up.

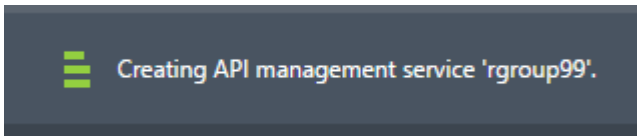
1. In the Azure portal, click **+ New**, select **Web + Mobile** and, in the list that appears, find and select **API Management**. (Note, **not** API App.)
2. This will open the old Azure management web site; the new portal does not yet support API Management. It will show the **Create an API Management Service** dialog.



3. In the **URL** box, enter a name, which must be globally unique. (Although the field label says **URL** this isn't the whole URL; it's just the first part of the domain name, so do not include the http:// part.) This must also be globally unique. Azure will tell you if you pick a name that is already in use by another API Management instance.
4. Set the **REGION** to the same one you've been using in previous walkthroughs.
5. Click the arrow at the bottom right. The second page of the dialog appears:



6. The **ORGANIZATION NAME** will be used on the Developer Portal that Azure creates for your API. You can use whatever name you like. Enter your email as the **ADMINISTRATOR E-MAIL**.
7. Click the tick at the bottom right.
8. A progress indicator will appear:



9. After a short while, Azure will hide this, but creation will continue in the background. You can bring back the progress indicator by clicking on the notification icon at the bottom right corner of the web page.

10. As noted, it will take Azure about half an hour to create the API Management instance. Once it has finished, move on to the next exercise.

Exercise 2: Wrap your API with API Management

In this exercise, you will wrap the API your Web App provides in the API Management instance you just created in Azure.

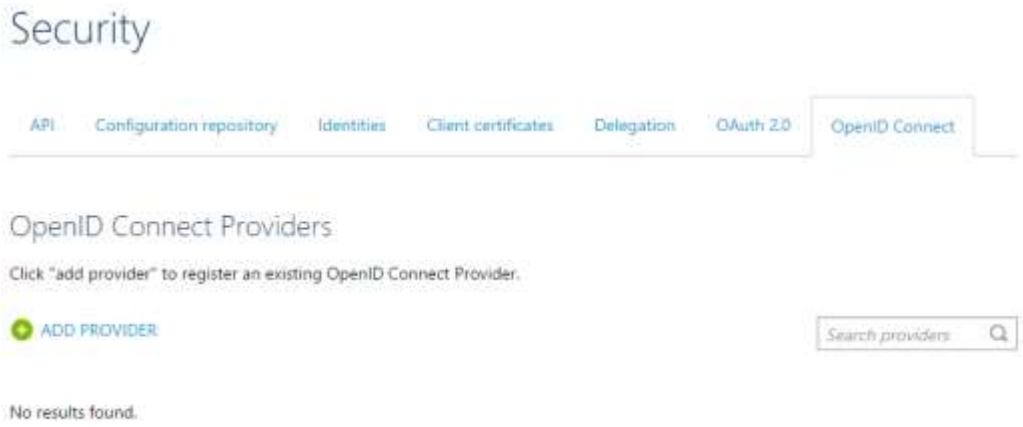
1. In the **API Management** section of the old Azure management site (https://manage.windowsazure.com/), find the API Management instance you selected and click its name.
2. At the bottom of the page, a **MANAGE** button will appear. Click it.

3. This will open the Dashboard page of the publisher portal for your new API Management instance in a new browser tab. Near the top of this page is an **IMPORT API** link. Click it.

4. This will display an **Import API** dialog. Click on **From URL**. Then, in the URL, paste in the link to the Swagger metadata for your API. You can find this by going to the new Azure portal in another tab, finding your Web App, going into the **Settings**, and choosing **API definition**. The URL you pasted there in an earlier lab will still be visible; that is the one to use.
5. Under **Specification format**, select **Swagger**.
6. Ensure **New API** is selected.
7. In **Web API URL suffix** type **rg**.
8. Ensure **HTTP** is not checked, and that **HTTPS** is checked.

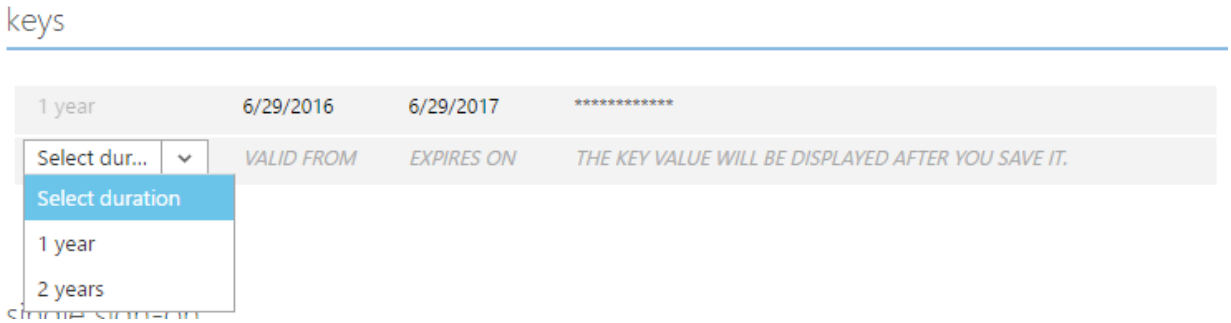
9. Click on **Products** and select **Starter** from the list.
10. It should look roughly like this (but with the URL being whatever is correct for your web app):

11. Click **Save**.
12. Although most of the default configuration for a new API will work for you, there's one change required: you need to configure how authentication works for this API. In the list on the left, click **Security**.
13. The web page shows several tabs, one of which is labeled **OpenID Connect**. Click on that tab header.



14. Click **ADD PROVIDER**.
15. In **Name**, type **rGroup AAD OpenID Connect**.
16. In **Description**, type **OpenID Connect login via AAD for the rGroup Web API**.

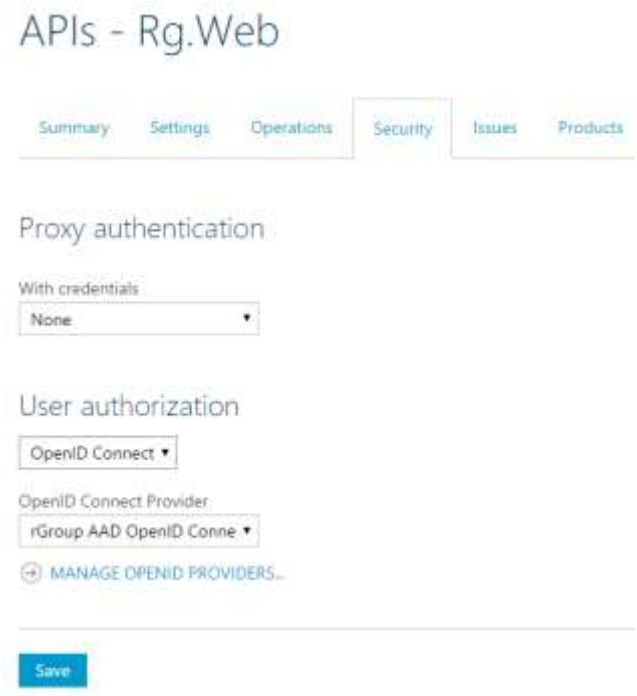
17. In **Metadata endpoint URL**, type a URL of this form https://login.microsoftonline.com/**YOUR AAD TENANT ID**/.well-known/openid-configuration but substitute your actual AAD Tenant ID. (You noted this down in the 'Getting Started' walkthrough. You should also be able to find it in the **Rg.Web** project's **Web.config** file on line 19.)
18. In **Client ID**, type in your AAD Application's client ID. (You also noted this down in the 'Getting Started' walkthrough, and it's also in the **Web.config** right after the Tenant ID.)
19. Next, you need to get a value for the **Client Secret**. For this, you'll need to go to the **ACTIVE DIRECTORY** section of the old Azure management site, find the **rGroup** AAD Directory you're using, find the **rGroup** application, and open its **CONFIGURE** page.
20. In the **keys** section, click on the drop-down on the last row in the list.



21. Select **1 year**.
22. Near the bottom of the web page, click **SAVE**.

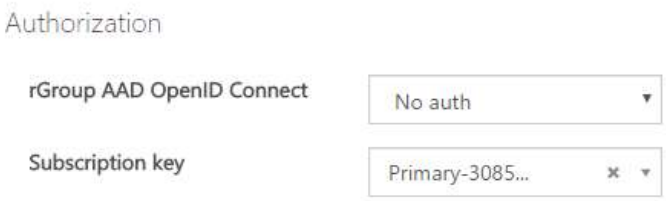
23. After a few seconds, a new key will appear in **keys** list. Copy the key, leave this web page open, return to the **API Management** portal, and paste that in as the **Client secret**.
24. Beneath the label reading **This is what the redirect_uri for authorization code grant type looks like:**, there is a URL. Copy that URL. Return to the AAD Application configuration page and paste this in as a new entry in the **REPLY URL** list in the **single sign-on** section.
25. Click **SAVE** again.
26. Return to the **API Management** portal and click **Save**.
27. In the list on the left, click **APIs**. It should show your **Rg.Web** API and also an **Echo API** that gets added for test and illustration purposes. Click on **Rg.Web**.

28. In the **APIs - Rg.Web** page that opens, click on the **Security** tab. Under **User authorization**, select **OpenID Connect**. The **OpenID Connect Provider** should appear, and it should already have selected your **rGroup AAD OpenID Connect** provider.



29. Click **Save**.
30. At the top right of the web page, Ctrl+click **Developer portal**.
31. A new browser tab will open showing the developer portal for your API. It will be showing a welcome message asking you to sign up. You can ignore this. Just click on the **APIS** link above that message.
32. The **Echo API** should be shown, along with your **Rg.Web** API. Click **Rg.Web**.
33. A list of all the operations provided by your API appears. Select **Test_Get** from the list. Click the **Try it** button that appears.

34. A UI appears on the web page enabling you to invoke the API. Part way down will be an **Authorization** section.



35. In the **rGroup AAD OpenID Connect** dropdown, select **Authorization code**. A dialog may appear to log you into AAD in order to obtain an access token. This will typically appear and then vanish as it decides you're already logged in and that it doesn't need you to enter any credentials.
36. At the bottom of the web page, click **Send**.
37. You should see information about the response appearing. The **Response status** should be **200 OK**, and the **Response content** should list a set of HTTP headers followed by a JSON array, showing all the claims visible to the service. We used this same endpoint to verify that authentication was working in the 'API App' lab. The purpose of this Test endpoint is to let us check that authenticated access is working as expected. You should see your name and email address in here, along with various other bits of information.
38. The API is now ready to use, but you need to do one more thing: API Management requires each developer to obtain a 'subscription key' which must be passed as part of each request. You will have automatically been assigned such a key when you created the API. (You can use the Publisher Portal to grant other developers access to the API If they sign up, they will get their own keys.) You'll need to take a copy of this key to be able to complete the next exercise.
39. Normally you'd look your keys up in the Profile page of the Developer Portal (which you can get to from the drop-down list at the top right hand corner of the web page), but it's already present on the page you're on right now. If you look at the **HTTP request**, which shows what the portal sent to the API, you can see some headers whose values have been obscured by a series of dots. You can view them by clicking the eye icon at the right-hand side of the request text box.

40. When you click this, the obscured headers become visible. Make a copy of the value of the **Ocp-Apim-Subscription-Key** header. Save this somewhere (e.g., in a text editor).
41. You are now ready to proceed to the next exercise.

Exercise 3: Use the API Management Endpoint

In this exercise, you will modify the client app to use the endpoint API Management has wrapped around your Web API.

1. In Visual Studio, in the **Rg.ClientApp** project, open **MainPage.xaml.cs**.

   Find the **CreateApiClientIfCredentialsAvailable** method and locate the line that initializes the **_apiClient** field. It will be at or near line 86 and will look something like this, but the name of the type being constructed will be different because it's based on the name of your Web App.

   ```
   _apiClient = new YourWebAppName(new Uri(WebAppUrl));
   ```

2. Replace the **WebAppUrl** text with **https://YOUR API INSTANCE.azure-api.net/rg/**, substituting the name of the API Management Instance you created earlier.
3. After this, add the following line of code, but replace the final argument with the value of the **Ocp-Apim-Subscription-Key** header that you copied at the end of the preceding exercise:

   ```
   _apiClient.HttpClient.DefaultRequestHeaders.Add(
       "Ocp-Apim-Subscription-Key",
       "22e1c0bd8311472ba392ff06fdb77d74");
   ```

4. Find the **getAlbumsButton_Click** method. It should still have a breakpoint from an earlier walkthrough. If it does not, set a breakpoint by clicking on the first line of code in the method then pressing **F9**.
5. Right-click on **Rg.ClientApp** in **Solution Explorer** and select **Set As Startup**, then press **F5** to run the application.
6. Click the **Log in** button and log in.
7. Click the **Get Albums** button.
8. Visual Studio should break when execution reaches the breakpoint. Press **F10** to step over the line of code that executes the test endpoint. Verify that the response returns a set of claims.
9. Step over the line that fetches your albums. Verify that it returns a list of your photo albums.
10. This verifies that by simply changing the code to use the base URL for the API Management wrapper for the endpoint and adding the subscription key header, you can use the API through API Management just as you did when using it directly.
11. Remove the breakpoint from the **getAlbumsButton_Click** method. Click **Get Albums** several times in a row. Eventually, you will get an **HttpOperationException** causing Visual Studio to break it. If you click **View Detail** on the exception dialog and look at the exception's **Response** property, you'll see the reason is that the API returned an HTTP 429 status with a message of 'Too Many Requests.' This is API Management's usage throttling in action. By default, some very conservative rate limits are in place, but you can configure more generous ones.

Exercise 4: Inspect the API Usage Information

In this exercise, you use the Publisher Portal's dashboard and analytics feature to see information about how the API has been used.

1. Return to the web browser tab that has the API Publisher Portal. If you don't have it open anymore, you can open the old Azure management site at https://manage.windowsazure.com/ go to the **API MANAGEMENT** section, select your API Management instance, and click the **MANAGE** button at the bottom of the web page.
2. Click the **Dashboard** link in the list on the left. This shows you some usage information for your APIs.

> **Note**: There is a slight lag between requests being made and showing up in the monitoring and analytics, so you might find the numbers shown don't exactly match with what you've been doing.

3. By default, this shows the last week of activity. Since this is a new API, you may as well click on the **Today** link above the graph so that it only shows today's activity.
4. This graph reports the number of calls that have been made across this entire API Management instance, how many failed, how much bandwidth was consumed, and the average response time.
5. You can get a more detailed view by clicking on **Analytics** in the list on the left. In the page that opens, the **At a glance** view shows the same information you saw in the graph on the dashboard. It also shows usage counts broken down by 'Developer' (based on the subscription key passed in the request header), 'product' (API Management's name for a collection of APIs made available as a set), and also by the APIs and operations used.

   Let's review what you have done. You created a new API Management instance and configured it to make your Web App's API available. This required only a small change to the client, after which you had automatic usage throttling to prevent rogue clients from hogging server

resources, and reporting on usage. In addition to the Publisher Portal, which presents usage information and analytics and allows the APIs to be configured, you also get a Developer Portal, which provides application developers with a place they can see what operations your API offers and how they should authenticate, and even lets them try operations out without having to write any code.

Last Updated: August 10, 2016.